# Chatbot4QR: Interactive Query Refinement for Technical Question Retrieval

Neng Zhang, Qiao Huang, Xin Xia, Ying Zou, David Lo, Zhenchang Xing

**Abstract**—Technical Q&A sites (e.g., Stack Overflow (SO)) are important resources for developers to search for knowledge about technical problems. Search engines provided in Q&A sites and information retrieval approaches (e.g., word embedding-based) have limited capabilities to retrieve relevant questions when queries are imprecisely specified, such as missing important technical details (e.g., the user's preferred programming languages). Although many automatic query expansion approaches have been proposed to improve the quality of queries by expanding queries with relevant terms, the information missed in a query is not identified. Moreover, without user involvement, the existing query expansion approaches may introduce unexpected terms and lead to undesired results.
In this paper, we propose an interactive query refinement approach for question retrieval, named **Chatbot4QR**, which can assist users in recognizing and clarifying technical details missed in queries and thus retrieve more relevant questions for users. Chatbot4QR automatically detects missing technical details in a query and generates several clarification questions (CQs) to interact with the user to capture their overlooked technical details. To ensure the accuracy of CQs, we design a heuristic-based approach for CQ generation after building two kinds of technical knowledge bases: a manually categorized result of 1,841 technical tags in SO and the multiple version-frequency information of the tags.
We develop a Chatbot4QR prototype that uses 1.88 million SO questions as the repository for question retrieval. To evaluate Chatbot4QR, we conduct six user studies with 25 participants on 50 experimental queries. The results are as follows. (1) On average 60.8% of the CQs generated for a query are useful for helping the participants recognize missing technical details. (2) Chatbot4QR can rapidly respond to the participants after receiving a query within approximately 1.3 seconds. (3) The refined queries contribute to retrieving more relevant SO questions than nine baseline approaches. For more than 70% of the participants who have preferred techniques on the query tasks, Chatbot4QR significantly outperforms the state-of-the-art word embedding-based retrieval approach with an improvement of at least 54.6% in terms of two measurements: Pre@k and NDCG@k. (4) For 48%-88% of the assigned query tasks, the participants obtain more desired results after interacting with Chatbot4QR than directly searching from Web search engines (e.g., the SO search engine and Google) using the original queries.

**Index Terms**—Interactive Query Refinement, Chatbot, Question Retrieval, Stack Overflow

## 1 Introduction

ONLINE technical Q&A sites, e.g., Stack Overflow[1] (SO) have emerged to serve as an open platform for knowledge sharing and acquisition [1], [2], [3]. The Q&A sites allow users to ask technical questions or provide answers to questions asked by others. For example, SO, which has been gaining increasing popularity in the software programming domain, has accumulated more than 19 million questions and 28 million answers as of December 20, 2019[2]. The questions and answers in the Q&A sites form a huge resource pool for developers to search for and solve programming problems [4], [5].

Question retrieval is a key step for users to seek for knowledge from Q&A sites, as well as a requisite step for many automatic tasks, such as answer summarization [6], API recommendation [5], and code search [7]. Most of the Q&A sites provide a search engine for users to retrieve questions using a query. Typically, a query is simply a free form text that describes a technical problem [8]. The search engines mainly rely on traditional information retrieval (IR) techniques (e.g., keyword matching and term frequency-inverse document frequency (TF-IDF) [9]), which cannot retrieve semantically similar questions for queries due to the lexical gaps between questions and queries [5]. Recently, word embedding techniques (e.g., word2vec [10]) are widely used by the state-of-the-art question retrieval approaches to bridge the lexical gaps [3], [5], [6], [11]. Such word embedding-based approaches have shown to be able to achieve better performance than traditional IR techniques.

A practical issue overlooked by the existing search engines and question retrieval approaches is that *it is not*

- *Neng Zhang is with the College of Computer Science and Technology, Zhejiang University, Hangzhou, China and Ningbo Research Institute, Zhejiang University, Ningbo, China and PengCheng Laboratory, Shenzhen, China.*
  *Email: nengzhang@zju.edu.cn*
- *Qiao Huang is with the the College of Computer Science and Technology, Zhejiang University, China.*
  *E-mail: tkdsheep@zju.edu.cn*
- *Xin Xia is with the Faculty of Information Technology, Monash University, Australia.*
  *E-mail: xin.xia@monash.edu*
- *Ying Zou is with the Department of Electrical and Computer Engineering, Queen's University, Canada.*
  *E-mail: ying.zou@queensu.ca*
- *David Lo is with the School of Information Systems, Singapore Management University, Singapore.*
  *E-mail: davidlo@smu.edu.sg*
- *Zhenchang Xing is with the Research School of Computer Science Australian National University, Australia.*
  *E-mail: zhenchang.xing@anu.edu.au*
- *Xin Xia is the corresponding author.*

1. https://stackoverflow.com/

2. https://data.stackexchange.com/

*an easy task for users to formulate a good query* [8], [12]. A survey conducted by Xu et al. [6] with 72 developers in two IT companies shows that a query could be imprecisely specified as users may not know the important keywords that the search engines expect. Rahman et al. [13] conducted an empirical study on code search using Google, which reveals that it is common for developers to miss some important technical details (e.g., programming languages and operating systems) in the initial queries. Consequently, *inaccurate queries will lead to unsatisfactory results of question retrieval*, as illustrated in the motivating example (see Section 2). To enhance the quality of queries, many automatic query expansion approaches have been proposed to expand queries with relevant terms extracted from a thesaurus (e.g., WordNet [14]) or similar resources [8], [12], [15]. Although the approaches can help retrieve relevant results, they are insufficient to obtain accurate results due to two reasons: (1) lack of techniques to identify the missing information in a query; and (2) queries expanded with unexpected terms without user involvement (as demonstrated in Section 6.1).

In this paper, we propose to interactively refine queries with users using a chatbot, named Chatbot4QR, in order to retrieve accurate technical questions from SO (or other Q&A sites) for users. Chatbot4QR focuses on accurately detecting the missing technical details in a query. It first retrieves an initial set of top-$n$ SO questions similar to the query. To build a responsive chatbot, we adopt a two-phase approach to explore a large-scale repository of SO questions by combining Lucene [16] (an ultra-fast text search engine that implements BM25) and a word embedding-based approach. Next, several **clarification questions** (CQs)[3] [17] are generated using a heuristic-based approach based on the technical SO tags appearing in the query and the top-$n$ similar questions. To identify the types of technical details missed in a query for CQ generation, we build two technical knowledge bases: a manually categorized result of 1,841 SO tags and the multiple version-frequency information of the tags. Then, Chatbot4QR interacts with the user by prompting the CQs to the user and gathers the user's feedback (i.e., a set of technical tags and versions answered by the user to CQs). Finally, the user's feedback is used to adjust the initial similarities of SO questions (by assigning a weight coefficient $\eta$ to the feedback), which results in a refined list of top-$k$ similar questions for recommendation.

To evaluate Chatbot4QR, we collected 1,880,269 SO questions as a large-scale repository for implementing a Chatbot4QR prototype and testing the performance of question retrieval for queries. Since our evaluation process contains six user studies that require a great amount of manual efforts, we built 50 experimental queries from the titles of another 50 SO questions. We conducted the user studies with 25 recruited participants to investigate the following research questions:

**RQ1. What are the proper settings of the parameters $n$ and $\eta$ in Chatbot4QR?**

In Chatbot4QR, there are two key parameters: (1) $n$ is the number of the initial top similar SO questions used for CQ generation; and (2) $\eta$ is the weight coefficient used for similarity adjustment of SO questions. We conducted a user study to evaluate the quality of CQs generated for queries by setting $n$ from 5 to 50 and the quality of the top ten SO questions recommended by setting $\eta$ from 0 to 1. Based on the results, we determine the proper settings of $n$ and $\eta$ as 15 and 0.2, respectively.

**RQ2. How effective can Chatbot4QR generate CQs?**

We conducted a user study to examine the usefulness of the CQs, i.e., whether the CQs can help the participants recognize the missing technical details in queries. The results show that on average, 60.8% of the generated CQs are useful for a query.

**RQ3. Can Chatbot4QR retrieve more relevant SO questions than the state-of-the-art question retrieval and query expansion approaches?**

We conducted a user study to evaluate the relevance of the top ten SO questions retrieved by Chatbot4QR and nine baselines, which apply two popular retrieval approaches (i.e., the Lucene search engine and a word embedding-based approach) and four query expansion approaches (see Section 5.3). The results show that Chatbot4QR outperforms the baselines by at least 54.6% in terms of two popular metrics: Pre@k and NDCG@k. For more than 70% of the participants, the improvement of Chatbot4QR over the baselines is statistically significant.

**RQ4. How efficient is Chatbot4QR?**

We recorded the execution time of Chatbot4QR during the experiments. Chatbot4QR takes approximately 1.3 seconds to start interaction with the user after receiving a query, which is efficient for practical uses.

**RQ5. Can Chatbot4QR help obtain better results than using Web search engines alone?**

We conducted four user studies (including the user study conducted in RQ3) for answering this research question. We asked the participants to search for their satisfied results for queries using Web search engines (e.g., the SO search engine and Google [18]) by applying the original queries and the refined queries after interacting with Chatbot4QR. Then, the participants evaluated the relevance of the search results. Finally, the participants chose their preferred results from three kinds of results: the two kinds of Web search results and the SO questions retrieved by Chatbot4QR. The results show that for 48%-88% of the assigned query tasks, the participants obtain more desired results either from Chatbot4QR or by applying the queries reformulated after the interaction with Chatbot4QR to Web search engines.

**Paper Contributions:**

1. We propose a novel chatbot to assist users in refining queries. To the best of our knowledge, this is the first work that uses an interactive approach to improving the quality of queries for technical question retrieval.
2. We conduct six user studies to evaluate Chatbot4QR. The evaluation results show that Chatbot4QR can generate useful CQs to help users recognize and clarify the missing technical details in queries efficiently. The refined queries contribute to retrieving better results than using the existing question retrieval approaches and Web search engines alone.

3. We define "clarification question" as a question that asks for some unclear information that is not given in the context of a query.

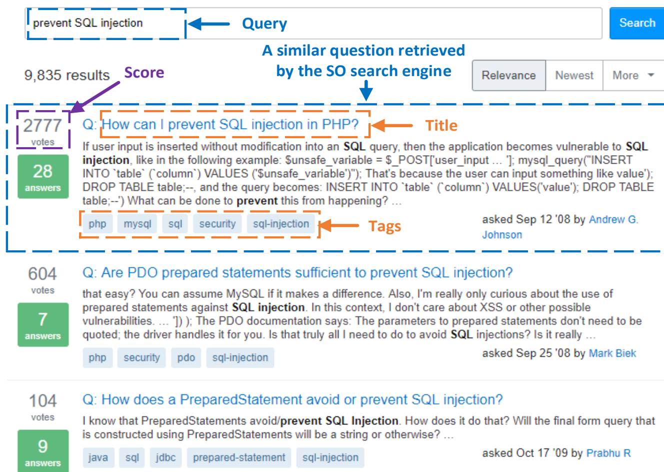


Fig. 1. The top three questions retrieved for a query by the SO search engine.

3. We release the source code of Chatbot4QR and the experimental dataset [19] to help other researchers replicate our experiments and extend our study.

**Paper Organization.** Section 2 describes a motivating example for our interactive query refinement approach. Section 3 presents the details of Chatbot4QR. Sections 4 and 5 report the experimental setup and results, respectively. Section 6 discusses several key aspects of Chatbot4QR and the threats to validity of our work. Section 7 reviews the related work. Section 8 concludes the paper and discusses future work.

## 2 Motivating Example

To motivate the use of an interactive approach to assisting users in refining queries, we illustrate the impact of a vague query on the quality of the questions retrieved by the SO search engine, and explain the key idea of Chatbot4QR.

Figure 1 shows an annotated screenshot of the top three questions retrieved by the SO search engine for a query "*prevent SQL injection*". Each retrieved question is tagged with a set of relevant technical terms, i.e., tags. For example, the first question is tagged with techniques, such as '*php*' and '*mysql*'. Obviously, the query is vague due to missing some important technical details, e.g., the preferred programming languages and databases. Looking at the tags associated with each question, the first and the second questions are related to the programming language '*php*', while the third question is related to '*java*'. Only the first question is explicitly tagged with the database '*mysql*'. Although the titles of the three questions are similar to the query, they are not satisfactory to every potential user as the users may have different technical background or programming context. For example, if a user prefers '*java*', the top two questions are undesirable, while the third question may be suitable depending on the user's preferred database. If a user is only familiar with the programming language '*c#*', none of the three questions is relevant to the user. However, we find that there are similar questions tagged with '*c#*' outside the top three returned results. To retrieve more desired questions for a user, it is necessary to assist the user in clarifying technical details that are not initially specified.

We propose Chatbot4QR to work interactively with users to improve the quality of queries. Chatbot4QR can heuristically generate several CQs to ask for two kinds of technical details: (1) the types of techniques widely adopted in software development, such as programming languages, databases, and libaries; and (2) the version of a technique as different versions of the technique may have substantial changes (e.g., '*python-2.x*' and '*python-3.7*'), which may cause version-sensitive problems. In Fig. 1, there are two programming languages in the top three retrieved questions, but no programming language is specified in the query. Therefore, a CQ can be generated, e.g., "*What programming language, e.g., php or java, does your problem refer to?*". Suppose that the user answers the CQ with '*java*', since '*java*' can have tags with multiple versions (e.g., '*java-7*' and '*java-8*'), a new CQ is generated to ask for a specific version, e.g., "*Can you specify the version of 'java', e.g., 7 or 8?*".

We strive to make our generated CQs easy for users to understand and answer, for the purpose of adoption in practice. Although a user needs to interact with our chatbot to answer the CQs, the amount of time spent is acceptable by the participants in our user studies (see Section 5.4). The feedback to CQs can help retrieve more relevant results and reduce the time required for the manual examination of undesirable results.

## 3 The Approach

Figure 2 gives an overview of our approach, which consists of two components: (1) *offline processing* which builds the Lucene index of SO questions, two language models (i.e., word2vec and word Inverse Document Frequency (IDF) vocabulary), and the categorization and version-frequency information of SO tags; and (2) *Chatbot4QR* which contains four main steps, namely ① retrieving the initial top-$n$ similar SO questions for a query, ② generating CQs by detecting the missing technical details in the query, ③ interacting with the user by asking the CQs to assist them in refining the query, and ④ producing the final top-$k$ recommended questions by adjusting the similarities of questions based on the user's feedback to CQs.

### 3.1 Offline Processing

As shown in Fig. 2, Chatbot4QR needs to retrieve the initial top-$n$ similar SO questions for a query before generating CQs. We build two language models, i.e., word2vec and word IDF vocabulary, to measure similarities between SO questions and queries, similar to the previous work [5], [6], [20]. The word2vec model is used to measure the semantic similarities among words; and the word IDF vocabulary measures the importance of words in the corpus. However, it is time-consuming to compute the semantic similarity between a query and each question in a large-scale repository, e.g., SO. To reduce the search space, we utilize Lucene to build the index for SO questions and retrieve a set of possibly similar questions before applying the word embedding-based approach. Moreover, we build two technical knowledge bases from SO tags for SO generation, i.e., the categorization and multiple version-frequency information of tags.

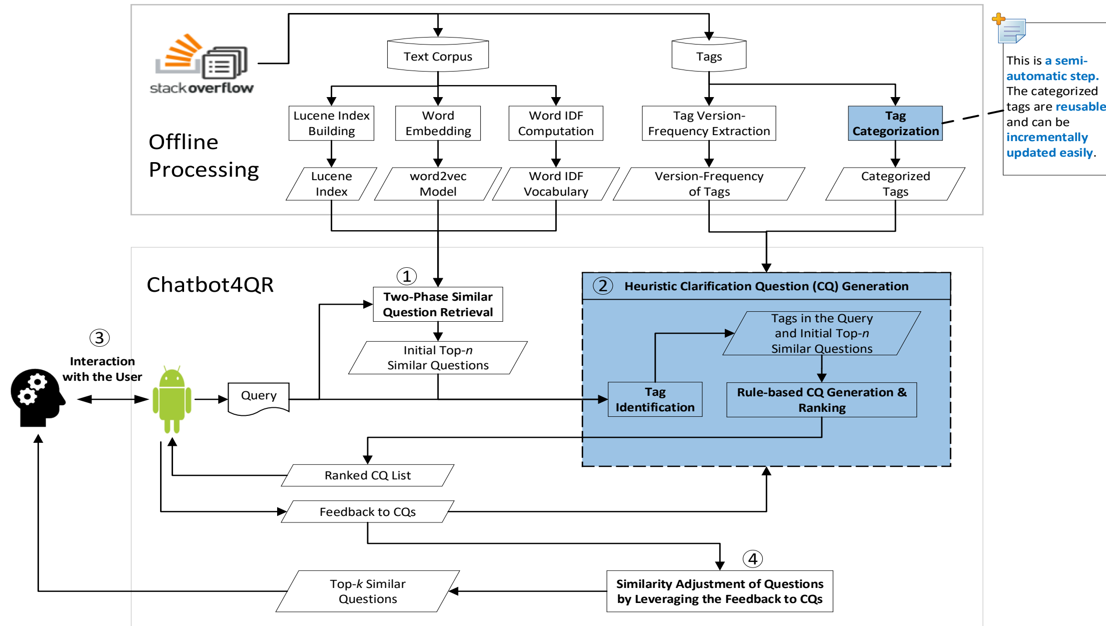


Fig. 2. An overview of our approach.

Questions tagged [java]

Description of *java*

Ask Question

Java (not to be confused with JavaScript, JScript or JS) is a general-purpose, platform-independent, statically typed, object-oriented programming language designed to be used in conjunction with the Java Virtual Machine (JVM). "Java platform" is the name for a computing system that has installed tools for developing and running Java programs. Use this tag for questions referring to the Java programming language or Java platform tools.

Unwatch Tag | Ignore Tag | Learn more… | Improve tag info | Top users | Synonyms (10) | java jobs

Noun phrase that indicates the type of *java*

1,621,547 questions

Newest | Active | Bountied 26 | Unanswered | More | Filter

Frequency of *java*

Tag synonyms for java

Incorrectly tagged questions are hard to find and answer. If you know of **common, alternate spellings or phrasings** for this tag, add them here so we can automatically correct them in the future. For example, suggest "bike" as a synonym for bicycle, or "sock" for socks.

The following tags will be remapped to java

java-se | .java | j2se | core-java | jdk | jre | java-libraries | oraclejdk | openjdk | javax

Ten synonyms of *java*

Fig. 3. An example SO tag "*java*".

We use the text corpus of SO questions (including the titles, tags, and bodies) and the SO tags (including the descriptions and synonyms crawled from the SO TagWiki [21]) as the input of the offline processing component. Figure 3 shows the description and ten synonyms of SO tag '*java*'. For questions, we remove the long code snippets enclosed in HTML tag $\langle pre \rangle$ from the bodies. We also reduce each word to its root form (aka. stemming) using the Porter stemmer in NLTK [22], a Python toolkit for natural language processing. As typical users would decide the relevance of a SO question to a query using the title and tags before checking the long body, we only consider the titles and tags of SO questions for question retrieval.

### 3.1.1 Lucene Index Building

We create a document for each SO question by gathering the title and tags, and build the index for all questions using Lucene.

### 3.1.2 Word Embedding

We apply the sentence tokenizer in NTLK to the titles and bodies of SO questions. Using the collected sentences, we train a word2vec model using Gensim [23] with the default parameter setting.

### 3.1.3 Word IDF Computation

We remove the stopwords provided in NLTK from SO questions and build the word IDF vocabulary by computing the IDF metric of each word.

### 3.1.4 Tag Version-Frequency Extraction

Many SO tags have multiple versions due to the update of techniques; and each version has its own frequency. The frequency of a SO tag reflects the number of SO questions that have been tagged with it. For example, the tag '*java*' has several versions, e.g., '*7*' and '*8*'; and the frequencies of '*java-7*' and '*java-8*' are 2,861 and 18,302, respectively. We extract the multiple version-frequency information of SO tags for generating a particular kind of CQs that ask users to specify the version of a technique that they are interested in.

By examining the SO tags with versions, there are two common templates of a technique and the corresponding versions: (1) concatenate a technique and a version number by '-', e.g., '*java-8*' and '*python-3.x*'; and (2) append a version number to a technique, e.g., '*sqlite3*' and '*sslv2*'. We extract the version numbers in SO tags using regular expressions. The extracted versions and the corresponding frequencies of each tag $t$ are stored in a dictionary, denoted as $ver_freqs(t)$. For example, two elements in $ver_freqs(\text{'}java\text{'})$ are {'7': 2,861, '8': 18,302}.

TABLE 1
Twenty types of SO tags. For each value in the "*#Tags Categorized to the Type*" column, the first number is the total number of tags categorized to the corresponding type; and the second number in the parenthesis is the number of tag synonyms categorized to the type.

| Type | #Tags Categorized to the Type | Example Tags |
|---|---|---|
| Library | 418(36) | jquery, winforms, pandas, opencv, numpy |
| Framework | 285(83) | .net, node.js, hibernate, spring, twitter-bootstrap |
| Tool | 211(27) | maven, curl, gcc, ant, openssl |
| Class | 171(3) | uitableview, listview, , httprequest, imageview, applet |
| Programming Language | 96(31) | javascript, java, c#, python, c++ |
| non-OS System | 77(12) | wpf, git, svn, gps, hdfs |
| Platform | 74(14) | azure, github, amazon-ec, google-cloud-platform, ibm-cloud |
| Service | 65(2) | outlook, firebase-authentication, gmail, google-cloud-messaging, google-play-services |
| Technique | 64(2) | jsp, reflection, proxy, nlp, deep-learning |
| Database | 63(16) | mysql, sql-server, mongodb, oracle, neo4j |
| non-PL Language | 60(4) | css, sql, wsdl, plsql, sparql, xml |
| Operating System | 58(28) | android, ios, linux, windows, macos |
| Server | 55(17) | tomcat, nginx, websphere, weblogic, jboss |
| Format | 46(6) | json, xml, csv, pdf, jar |
| Plugin | 44(6) | silverlight, jquery-validate, android-gradle, pydev, jstree |
| Environment | 33(9) | eclipse, netbeans, visual-studio, webstorm, spyder, jdeveloper |
| Engine | 32(2) | apache-spark, google-app-engine, elasticsearch, andengine, innodb |
| Design Pattern | 15(0) | model-view-controller, singleton, adapter, inversion-of-control, decorator |
| Model/Algorithm | 15(1) | dom, classification, rsa, svm, logistic-regression |
| Browser | 15(6) | google-chrome, internet-explorer, firefox, safari, opera |
| **Total** | **1,841(305)** | |

### 3.1.5 Tag Categorization

Categorizing SO tags to a set of meaningful types is critical for generating CQs for queries. Existing work that categorizes SO tags (e.g., [24], [25], [26]) is either incomplete or too fine-grained for our CQ generation. For example, only six types of SO tags were considered by Ye et al. [24] while neglecting some important types, e.g., *operating system* and *plugin*. Chen et al. [25] automatically generated 167 types where the analogous types (e.g., *library* and *module*) should be better merged. Incomplete types result in missing useful CQs, while fine-grained types lead to redundant CQs.

We strive to manually build a high-quality categorization of SO tags. However, the manual categorization of more than 50 thousands tags in SO is a cumbersome task. As the chances for querying the low frequency tags are low, we focused on the tags with the frequency of more than 1,000. As a result, we selected 4,158 tags. Despite the synonyms defined in SO, we also considered the tags marked with version numbers to be synonyms. For example, an extended synonyms set is {'*java*', '*java-se*', '*jdk*', '*java-7*', '*java-8*', ...}. We kept only the most frequent tag in each set of synonyms. Consequently, we obtained 3,772 tags. Then, we categorized the tags by using two iterations of a card sorting approach [27] as follows.

- **Build a set of types.** We observed that many SO tags have a noun phrase in the first description sentence to indicate the types of them, which are typically expressed in the form of "*X is a/an noun phrase ...*" [26]. As shown in Fig. 3, the first description sentence of '*java*' shows that it is a programming language. We randomly sampled 349 tags from the 3,772 tags, which is a statistically significant sample size considering a confidence level of 95% and a confidence interval of 5. We used the Stanford POS (Part-of-Speech) tagger in NLTK to parse the first description sentence of each tag and extracted the first noun phrase behind the articles '*a*' or '*an*'. The first two co-authors independently examined the noun phrases and built their own sets of types. Then, the two co-authors and a postdoc (who is not a co-author of the paper) together discussed the disagreements, eventually resulting in 20 types, as presented in Table 1. The two types '*non-PL Language*' and '*non-OS System*' respectively represent the non-programming languages (e.g., the query language '*sql*') and the non-operating systems, e.g., the version-control system '*svn*'.
- **Categorize tags based on types.** Based on the built types, the two co-authors further independently categorized each of the 3,772 tags. In total, 1,641 tags were initially categorized by at least one co-author. The uncategorized tags belong to the ignored types which are too general and are likely to be useless for CQ generation, e.g., *concept* and *keyword*. There were 215 tags with disagreement. The Fleiss Kappa [28] value of the two categorization results is 0.86, meaning an almost perfect agreement. The two co-authors and the postdoc worked together again to discuss the disagreements. Finally, they reached consesus on the categorization of 1,548 tags. The synonyms of a tag were then categorized to be the same type(s) as the tag. Table 1 presents the numbers of tags categorized to each of the 20 types, along with example tags. The numbers in parentheses are the numbers of synonyms categorized to the corresponding types. For example, "*1,841(305)*" in the bottom row means that 1,841 tags including 305 synonyms are categorized to the 20 types. Note that the sum of the number of SO tags categorized to the 20 types is 1,897, which is larger than 1,841, since some tags are categorized to multiple types. For example, the tag '*xml*' is categorized to the two types '*non-PL Language*' and '*Format*'.

In our approach, tag categorization is a semi-automatic step. We took approximately 65 hours and nine hours to complete the two iterations, respectively. It is worth to mention that the categorized tags are reusable and can be incrementally updated easily. More specifically, when the frequencies of a number of (e.g., 50) uncategorized SO tags exceed 1,000, we can automatically extract the noun phrases from the first description sentences of the tags and then categorize them.

### 3.2 Chatbot4QR

Once the offline processing component is completed, the Chatbot4QR component shown in Fig. 2 is launched when a user submits a query. The query is processed first by two steps: stemming and stopword removal. Then, the four steps ① - ④ in Fig. 2 are conducted to help the user refine the query if it has unclear technical details and recommend the top-$k$ similar SO questions to the user.

#### 3.2.1 Two-Phase Similar Question Retrieval

To detect if there are technical details left out in the query, denoted as $q$, we obtain the initial top-$n$ SO questions similar to $q$ using a two-phase approach. More specifically, we first use the Lucene search engine to retrieve a reduced set of $N$ possible similar questions based on the Lucene index built for SO questions. Then, we use the word embedding-based approach adopted in the previous work [5], [6], [20] to retrieve the top-$n$ semantically similar questions, denoted as $iSimQ_n(q)$, from the reduced set. To ensure that the majority of semantically similar questions can be covered by the reduced set, we set $N$ = 10,000.

#### 3.2.2 Heuristic Clarification Question Generation

Based on the initial top-$n$ similar SO questions obtained for query $q$, we design a heuristic-based approach to automatically detecting the missing technical details in $q$ and generate a set of CQs to help the user refine $q$ interactively. The approach contains two sub-steps: tag indentification and rule-based CQ generation & ranking.

**Tag identification**. To generate CQs, we identify the SO tags appearing in $q$ and the top-$n$ similar questions $iSimQ_n(q)$. This is not an easy task due to the diverse appearances of SO tags in natural language texts. More specifically, every SO tag is lowercase and multiple tokens are concatenated by '-', e.g., '*sql-injection*'. Moreover, SO tags can have versions, e.g., '*java-8*'. In contrast, the tags and versions can appear in a variety of forms in queries and the titles of SO questions, e.g., '*java 8*', '*Java8*', and '*Java 8's*'. Before identifying tags in $q$ and the similar questions, we transform each categorized SO tag by removing the possible version and replacing '-' with a blank character. We also transform the original query as well as the original title and tags of each question in $iSimQ_n(q)$ by (1) converting them to lowercase, (2) replacing punctuations (except '#' and '+' as such symbols can be used as a part of a tag, e.g., '*c#*' and '*c++*') with a blank character, and (3) separating the possible version at the end of each token.

Using the transformed results described above, we identify the tags in $q$ and each question $Q \in iSimQ_n(q)$. We also extract the version number, if it exists, of each tag identified from $q$. We filter out the version numbers of tags in the top-$n$ similar questions as we directly use the version-frequency information of tags stored in $ver_freqs$ (see Section 3.1.4) to generate CQs, which may help cover more similar questions outside the top-$n$. We group the two sets of tags identified from $q$ and similar questions by the types of tags. The two grouped sets of tags are denoted as $typed_tags(q)$ and $typed_tags(iSimQ_n(q))$, respectively. Table 2 presents the grouped tags identified from the query and the top three SO questions shown in Fig. 1. In the table, we display the two most frequent versions of each tag in $typed_tags(iSimQ_3(q))$.

TABLE 2
Tags identified from the query "*prevent SQL injection*" ($q$) and the top three SO questions shown in Fig. 1.

| Type | typed_tags($q$) | typed_tags(iSimQ$_3$($q$)) |
|---|---|---|
| Programming Language | | { php: ['7', '5.3'], java: ['8', '7'] } |
| non-PL Language | { sql: '' } | { sql: [] } |
| Database | | { mysql: ['2', '5.7'] } |
| Framework | | { .net: ['4.0', '3.5'] } |
| Library | | { jdbc: [] } |
| Class | | { pdo: [] } |
| Technique | { sql-injection: '' } | { sql-injection: [] } |

**Rule-based CQ generation & ranking**. By comparing the two sets of identified tags, we generate three kinds of CQs for query $q$ using the following three heuristic rules:

- **Rule 1 (version related CQ generation).** For each tag $t$ in $typed_tags(q)$, if it has no specified version in $q$ and it is a multi-version tag (i.e., $len(ver_freqs(t)) \geq 2$), a `version` related CQ is generated, such as "*Can you specify the version of* $t$*, e.g.,* $v_1$ *or* $v_2$*?*". $v_1$ and $v_2$ are the two most frequent versions of $t$ in $ver_freqs(t)$, which are displayed to help the user better understand the CQ and provide feedback correctly.
- **Rule 2 (selection related CQ generation).** For each type $type$ in $typed_tags(iSimQ_n(q))$ but not in $typed_tags(q)$, if there are two or more tags included in the type, a `selection` related CQ is generated, such as "*What type, e.g.,* $t_1$ *or* $t_2$*, are you using?*". $t_1$ and $t_2$ are the two most frequent tags belonging to $type$ in $typed_tags(iSimQ_n(q))$. To make the selection related CQs sounded more natural, we customized the CQ expressions for the 20 types of SO tags, as shown in Table 3.
- **Rule 3 (confirmation related CQ generation).** For each type $type$ in $typed_tags(iSimQ_n(q))$ but not in $typed_tags(q)$, if only one tag $t$ is included in the type, a `confirmation` related CQ is generated, such as "*Are you using* $t$*? (y/n), or some other types.*".

**Rule 3** is a special case of **Rule 2**. We distinguish them because a confirmation related CQ is more informative, implying that only one tag belonging to that type is identified from the initial top-$n$ similar questions. If a user indeed uses the asked technique, they can easily answer the CQ with '*y*'.

In the subsequent interaction with the user, CQs that are more relevant to the query should be asked first. We rank the generated CQs by assigning a score to each CQ as follows:

- If *cq* is a version related CQ, its score is set to 1.0 because the tag asked in *cq* is explicitly specified by the user.
- If *cq* is a selection or confirmation related CQ, its score is calculated according to the similarities of the questions that contain any tags belonging to the *type* asked in *cq*, i.e., $\frac{\sum_{Q \in iSimQ(type)} sim(q,Q)}{\sum_{Q \in iSimQ_n(q)} sim(q,Q)}$, where $iSimQ(type)$ denotes the subset of questions in $iSimQ_n(q)$ that contains a tag categorized to *type*; and $sim(q, Q)$ is the semantic similarity between $q$ and $Q$.

TABLE 3
Customized CQ expressions for the 20 types of SO tags shown in Table 1. In each CQ expression, "*X*" and "*Y*" are two example SO tags of the corresponding type that appear in the initial top similar SO questions retrieved for a query.

| Type | Customized Selection Related CQ Expression for the Type |
|---|---|
| Library | Which library, e.g., X or Y, are you using? |
| Framework | If you are using a framework, e.g., X or Y, please specify: |
| Tool | Maybe you are using a tool, e.g., X or Y, for the problem. If so, what is it? |
| Class | Are you using a specific class, e.g., X or Y? Please input it: |
| Programming Language | What programming language, e.g., X or Y, does your problem refer to? |
| non-OS System | Apart from the operating system (OS), is there a non-OS, e.g., X or Y, used for your problem? |
| Platform | Tell me a possible platform, e.g., X or Y, you are using: |
| Service | For the problem, if you are using a service, e.g., X or Y, please provide: |
| Technique | Please give a possible technique, e.g., X or Y, you might use for the problem: |
| Database | I want to know whether you are using a database, e.g., X or Y. Can you provide it? |
| non-PL Language | Despite the programming language (PL), are you using any non-PL languages, e.g., X or Y? |
| Operating System | Could you provide an operating system, e.g., X or Y? |
| Server | Which server, e.g., X or Y, does your program intend to run on? |
| Format | What is the format, e.g., X or Y, of the data/file you are handling? |
| Plugin | I am wondering if you are using a plugin, e.g., X or Y. Specify it if there is one: |
| Environment | Would you like to provide an environment, e.g., X or Y, you are using? |
| Engine | Give me a possible engine, e.g., X or Y, that you need to execute your program: |
| Design Pattern | Any design patterns, e.g., X or Y, used for your problem? |
| Model/Algorithm | Do you use a model or an algorithm, e.g., X or Y? Please specify: |
| Browser | Your problem may be related to a browser, e.g., X or Y. Can you specify it? |

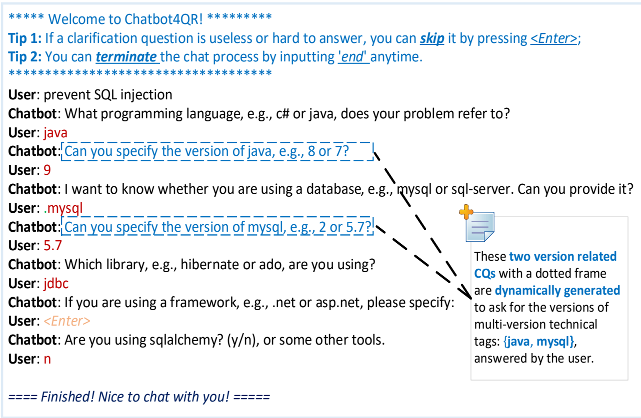


Fig. 4. The interaction with a user for the query shown in Fig. 1.

### 3.2.3 Interaction with the User

Based on the ranked CQ list, Chatbot4QR interacts with the user by asking each CQ one-by-one and gathers the user's feedback. Figure 4 illustrates the chat process with a user by submitting the query shown in Fig. 1 to Chatbot4QR. The five CQs without a dotted frame are initially generated based on the top 15 (the proper value of the parameter $n$ in Chatbot4QR, as evaluated in Section 5.1) similar SO questions retrieved using the two-phase approach. As shown in Fig. 4, Chatbot4QR has the following features:

1. It can interact with the user in multiple rounds.
2. It can generate new version related CQs to ask for the versions of the multi-version tags (e.g., '*java*' and '*mysql*') that are answered by the user to confirmation or selection related CQs.
3. To be user-friendly, it allows the user to skip any CQs that might be not useful or difficult to answer by pressing $\langle Enter \rangle$, or to terminate the interaction anytime.

### 3.2.4 Similarity Adjustment of Questions

We distinguish two kinds of a user's feedback to the CQs of query $q$: (1) **positive feedback**, denoted as $pfb(q)$, which includes the tags and versions answered by the user; and (2) **negative feedback**, denoted as $nfb(q)$, which includes the tags involved in the confirmation related CQs whose answers are explicitly '$n$' (means that the user does not use the asked technique). We do not consider the possible negative feedback to CQs since the user's rationale is unknown. For example, if a confirmation related CQ has no answer (i.e., the CQ was skipped), it is not certain that the user does not use the asked technique. It might be the reason that users are not familiar with the programming context and thus have difficulties in answering. In Fig. 4, the positive and negative feedback given by the user are

- $pfb(q)$ = {'*java 9*', '*mysql 5.7*', '*jdbc*'},
- $nfb(q)$ = {'*sqlalchemy*'}.

Using the two kinds of feedback, we adjust the semantic similarity between $q$ and each question $Q$ in the reduced set retrieved using Lucene as

$$sim(q,Q) = \quad sim(q,Q) \times (1 + \eta \times (\textstyle\sum_{e \in pfb(q)} md(e,Q) - \sum_{e \in nfb(q)} md(e,Q))) \tag{1}$$

where $md(e,Q)$ measures the degree that $Q$ matches the tag and its possible version in the feedback element $e = (t,v)$ (where $t$ is the tag and $v$ is the version), e.g., '*java 9*'. The coefficient $\eta \in [0,1]$ is used to weight the importance of the technical feedback. A larger $\eta$ means to put more weight on the feedback. More specifically, $\eta = 0$ ignores the feedback, while $\eta = 1$ means that the feedback has the same importance as the original query. In this work, we define $md(e,Q)$ as

$$md(e,Q) = \begin{cases} 1.5, & \text{if } e.v \text{ exists and both} \\ & e.t \text{ and } e.v \text{ are matched by } Q \\ 1.0, & \text{if only } e.t \text{ is matched by } Q \\ 0, & \text{otherwise.} \end{cases} \tag{2}$$

The idea of Eq. 1 is to increase (resp. decrease) the semantic similarity of $Q$ according to the amount of positive (resp. negative) feedback matched by $Q$. A refined list of the

TABLE 4
Fifty experimental queries.

| No. | SO Question ID | Experimental Query (the Title of the SO Question) |
|---|---|---|
| 1 | 17294809 | Reading a line using scanf() |
| 2 | 423006 | How do I generate points that match a histogram? |
| 3 | 15389110 | How to convert Json String with dynamic fields to Object? |
| 4 | 2733356 | Killing thread after some specified time limit in Java |
| 5 | 20458401 | How to insert multiple rows into database using hibernate? |
| 6 | 15626686 | Better way to parse xml |
| 7 | 2592985 | ArrayList shallow copy iterate or clone() |
| 8 | 6262084 | how to slide image with finger touch in android? |
| 9 | 5108926 | how to encrypt data using AES in Java |
| 10 | 7918593 | How can I determine the week number of a certain date? |
| 11 | 90838 | How can I detect the encoding/codepage of a text file |
| 12 | 12981190 | How to make a static variable thread-safe |
| 13 | 22173762 | Check if two Lists are equal |
| 14 | 2411893 | Recognize numbers in images |
| 15 | 3561202 | Check If Instance Of A Type |
| 16 | 8702165 | How to clone (and restore) a DOM subtree |
| 17 | 11182924 | How to check if JavaScript object is JSON |
| 18 | 30950032 | How can I run multiple NPM scripts in parallel? |
| 19 | 531998 | Set path programatically |
| 20 | 28052395 | Find whether a 2d matrix is subset of another 2d matrix |
| 21 | 14268053 | Most efficient way to calculate pairwise similarity of 250k lists |
| 22 | 5450055 | How can I improve my INSERT statement performance? |
| 23 | 3548495 | Download, extract and read a gzip file in Python |
| 24 | 44274701 | Make predictions using a tensorflow graph from a keras model |
| 25 | 4869189 | How to transpose data in a csv file? |
| 26 | 215557 | Most elegant way to implement a circular list (FIFO) |
| 27 | 1558402 | Memory usage of current process in C |
| 28 | 1805518 | Replacing all non-alphanumeric characters with empty strings |
| 29 | 6390339 | How to query XML that has XSL in Java with XPath? |
| 30 | 2676719 | Calculating the angle between two points |
| 31 | 10975913 | How to make a new list with a property of an object which is in another list |
| 32 | 8892073 | how to compare webpages structure (dom) similarity in java? |
| 33 | 9963331 | java : How to know how many Threads have been Created and running? |
| 34 | 891345 | Get a screenshot of a specific application |
| 35 | 8910840 | Using LINQ to extract ints from a list of strings |
| 36 | 21461102 | Converting Html Table to JSON |
| 37 | 6773550 | Get id of div from its class name |
| 38 | 2617515 | Recommendation for a HTTP parsing library in C/C++ |
| 39 | 1323824 | how to read numbers from an ascii file (C++) |
| 40 | 3823921 | Convert big endian to little endian when reading from a binary file |
| 41 | 13340955 | Convert linear Array to 2D Matrix |
| 42 | 1623849 | Fastest way to zero out low values in array? |
| 43 | 32109319 | How to implement the ReLU function in Numpy |
| 44 | 14472795 | How do I sort a list of datetime or date objects? |
| 45 | 5741518 | Reading each column from csv file |
| 46 | 22722079 | Choosing elements from python list based on probability |
| 47 | 7891697 | Numpy Adding two vectors with different sizes |
| 48 | 8022530 | Python check for valid email address? |
| 49 | 459981 | BeautifulSoup - modifying all links in a piece of HTML? |
| 50 | 10052912 | How to sort dictionaries of objects by attribute value in python? |

top-$k$ similar questions is produced based on the adjusted similarities and recommended to the user.

# 4 Experimental Setup

Chatbot4QR is an interactive approach that considers users' personalized technical background and programming context to retrieve desired questions. We design a series of user studies to evaluate Chatbot4QR. In this section, we describe the experimental setup of our user studies. Our experimental environment is a laptop with Intel Core i5-8300H CPU, 16G RAM, and Windows 10 Operating System.

## 4.1 Data Collection and Prototype Implementation

We downloaded the official SO data dump released in September, 2018 and built a repository of 1,880,269 SO questions that are tagged with six popular programming language tags: {'*javascript*', '*java*', '*c#*', '*python*', '*c++*', '*c*'}. To ensure the quality of our repository, every question needs to have an accepted answer and a positive score (i.e., the votes of a question shown in Fig. 1). Using the collected questions, we built a text corpus by removing the long code snippets in the bodies of questions and processing all words in the title, tags, and body of each question using the Porter stemmer in NLTK. We then trained a word2vec model using Gensim (with the default parameter setting), computed the word IDF vocabulary, and built the Lucene index for all questions. Moreover, we crawled the descriptions and synonyms of 55,661 SO tags from the TagWiki, and built two technical knowledge bases: the categorization and version-frequency information of tags. The details of these offline steps are described in Section 3.1.

As described in Section 3.2, Chatbot4QR has three parameters: (1) $n$ is the number of the initial top similar SO questions used for CQ generation; (2) $\eta$ is the weight coefficient of users' technical feedback in Eq. 1 used to adjust the similarities of questions; and (3) $k$ is the number of the top similar questions recommended to the user. We determined the proper settings of $n$ and $\eta$ as 15 and 0.2, respectively, by conducting a user study (see Section 5.1). Considering the fact that users are likely to be only interested in the top ranked results [29], we set $k$ = 10 in our prototype implementation, similar to the previous work [30], [31], [32].

## 4.2 Experimental Query Selection

In the existing research work on information retrieval from SO [5], [6], [8], [20], [32], [33], the experimental queries used for evaluation are built from the titles of SO questions selected according to some criteria, of which two commonly used criteria are: (1) the questions should have accepted answers; and (2) the scores of questions should be higher than a threshold (e.g., 5). This is suitable because the title of a SO question is a simple text that briefly describes a technical problem that a developer wants help for. We built 50 experimental queries from the titles of SO questions outside our repository. We chose 50 queries due to two reasons: (1) it is a relatively common number of experimental queries used in the previous work [30], [32], [33]; and (2) our user studies contain six consecutive stages (see Fig. 5) which require a great amount of manual efforts.

Our experimental queries were selected as follows. We first collected the popular SO questions which are tagged with the aforementioned six programming languages but not in our repository using two criteria: (1) the view count should be no less than 1,000; and (2) the score should be at least five. Then, we randomly selected 50 queries from the titles of the collected questions. For each query, we further ensured that there is no duplicate question contained in the repository, similar to the previous work [6]. As listed in Table 4, the 50 queries cover a variety of technical problems, which involve different techniques, e.g., programming languages, databases, and deep learning libraries. Some of the queries are simple, e.g., "*Reading a line using scanf()*" while others are complex, e.g., "*How to sort dictionaries of objects by attribute value in python?*". Moreover, there are queries expressed with technical terms, e.g., "*Killing thread after some specified time limit in Java*", while some queries have no specified technique, e.g., "*Recognize numbers in images*". The diversity of the queries can improve the generality of our experiment results.

We processed the queries by performing stemming and stop word removal. Based on the Lucene index built for SO questions, we retrieved the top $N$=10,000 similar questions for each query using the Lucene search engine. We then re-ranked the 10,000 questions by measuring semantic similarities between the questions and the query using the word embedding-based approach adopted in the previous work [5], [6].

TABLE 5
Profiles of 25 participants.

| Participant | Familiar Programming Languages | #Years of Programming Experience |
|---|---|---|
| P1 | python | 3.5 |
| P2 | python | 4.0 |
| P3 | java, python | 8.0 |
| P4 | java, python | 6.0 |
| P5 | java | 4.5 |
| P6 | python | 7.5 |
| P7 | java, python | 4.0 |
| P8 | java, python, c | 10.0 |
| P9 | java, python | 5.5 |
| P10 | java | 3.5 |
| P11 | java | 3.0 |
| P12 | java, c# | 2.0 |
| P13 | java, python, matlab | 8.5 |
| P14 | java, python, c# | 6.5 |
| P15 | java | 3.5 |
| P16 | java, python | 4.0 |
| P17 | java, python, c++ | 8.0 |
| P18 | java, python | 8.5 |
| P19 | java, javascript | 2.5 |
| P20 | java | 3.5 |
| P21 | java, python | 8.0 |
| P22 | java, javascript | 7.0 |
| P23 | java, python | 11.0 |
| P24 | java, python | 6.5 |
| P25 | python, c, c++ | 9.0 |

### 4.3 Participant Recruitment

To conduct our user studies shown in Fig. 5 for evaluating Chatbot4QR, we recruited participants through the mailing lists of the first and the third co-authors' colleges. In the email, we briefly introduced Chatbot4QR and our evaluation plan, and asked a few questions about the programming background. We received 25 responses that agreed to join our user studies. The number of our participants is close to the numbers of participants used to conduct user studies in the previous work [34], [35], which is considered to be sufficient for our user studies. Table 5 presents the profiles of the 25 participants. Since some of the participants have working experience in companies like Hengtian[4], the "*#Years of Programming Experience*" column shows the years of both working experience and student experience in programming for each participant. We observe that the participants have diverse familiar programming languages. Some of them are only familiar with Java or Python, while others have multiple familiar languages. Moreover, there are notable differences in the participants' years of programming experience (from 2 to 11 years) with an average of 5.92 years.

We asked the participants to review the experimental queries and no participant reported being unable to understand the queries after we allowed them to search for the definitions of unfamiliar technical terms (e.g., '*LINQ*' and '*NPM*') online. As listed in Table 4, our queries cover multiple programming languages, e.g., Java, Python, and C++. We did not guarantee that the participants are familiar with all the programming languages because **Chatbot4QR intends to help both experienced developers and novices**.

4. Hengtian is an outsourcing company in China that has more than 2,000 employees and mainly does outsourcing projects for American and European corporations.

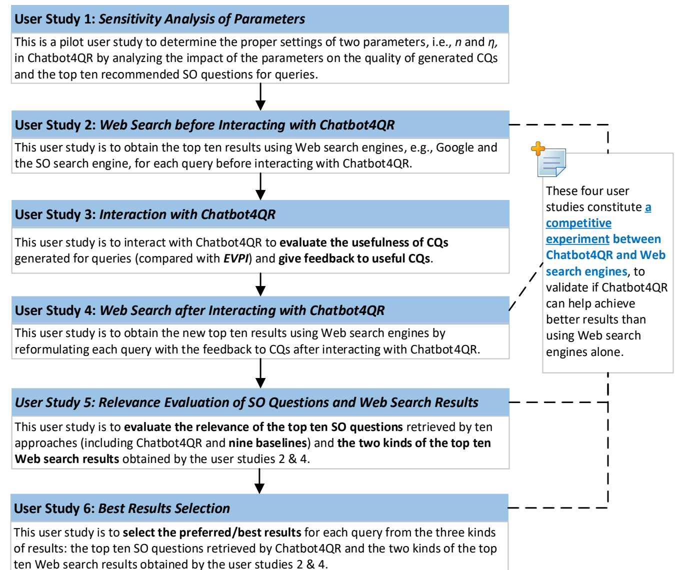


Fig. 5. The flow of our six user studies.

The diversity of the participants' technical background can help improve the generality of our experiment results.

### 4.4 Research Questions and the Allocation of Queries to Participants for User Studies

As shown in Fig. 5, we designed six user studies to investigate the following research questions:

**RQ1.** What are the proper settings of the parameters $n$ and $\eta$ in Chatbot4QR?

**RQ2.** How effective can Chatbot4QR generate CQs?

**RQ3.** Can Chatbot4QR retrieve more relevant SO questions than the state-of-the-art question retrieval and query expansion approaches?

**RQ4.** How efficient is Chatbot4QR?

**RQ5.** Can Chatbot4QR help obtain better results than using Web search engines alone?

It is a cumbersome task for a participant to conduct the five user studies 2-6 (the user study 1 is a pilot user study) for all the 50 experimental queries. Therefore, we allocated 25 queries to each participant as follows.

We randomly divided the 50 queries into two equally sized groups: **QG1** and **QG2**. The queries in QG1 are indexed by Q1-Q25 and those in QG2 are indexed by Q26-Q50, as shown in Table 4. From the 25 participants, we first randomly selected five participants, denoted as **PG0** = P1-P5, who are responsible for conducting a pilot user study to determine the proper settings of the two key parameters $n$ and $\eta$ in Chatbot4QR. Then, we divided the remaining 20 participants evenly into two groups: **PG1** = P6-P15 and **PG2** = P16-P25, while ensuring that members of the two groups have comparative years of programming experience.

Table 6 lists the allocation of queries to participants for our six user studies and the research questions investigated by each user study. More specifically, the **user study 1** is a pilot user study to investigate **RQ1** by randomly selecting ten queries and allocating the queries to the participants in PG0. For the other five user studies, we allocated QG1

TABLE 6
The allocation of queries to participants for the six user studies shown in Fig. 5; and the research questions investigated by each user study. "*RQ1-RQ5*" are the five research questions. "*PG0-PG2*" are three participant groups. "*Q1-Q50*" are the 50 experimental queries. "*QG1*" and "*QG2*" are two query groups.

| User Study No. | Investigated Research Questions | Allocation of Queries to Participants |
|---|---|---|
| 1 | RQ1 | Ten queries randomly selected from Q1-Q50 are allocated to **PG0** |
| 2 | RQ5 | **QG1** are allocated to **PG1**, **QG2** are allocated to **PG2** |
| 3 | RQ2, RQ4 | |
| 4 | RQ5 | |
| 5 | RQ3, RQ4, RQ5 | |
| 6 | RQ5 | |

and QG2 to PG1 and PG2, respectively. The **user study 3** investigates **RQ3** by examining the usefulness of the CQs generated by Chatbot4QR for the 50 queries. **RQ4** is answered by recording the amount of time spent on the steps of Chatbot4QR during the **user studies 3** and **5**. The **user studies 2**, **4**, **5**, and **6** constitute a competitive experiment to investigate **RQ5** by comparing the quality of the top ten SO questions retrieved by Chatbot4QR and the two kinds of the top ten results retrieved using Web search engines (e.g., the SO search engine and Google) before and after interacting with Chatbot4QR. As the participants interact more with Chatbot4QR, they may gradually learn to recognize some technical details missed in their initial queries. Therefore, we required the participants to perform the **user study 2** (i.e., *Web Search before Interacting with Chatbot4QR*) before the **user study 3** (i.e., *Interaction with Chatbot4QR*), in order to minimize the learning effect that the participants may transfer the knowledge learned from Chatbot4QR to enhance the queries for Web search.

Before performing the user studies, the participants are expected to find a solution for each allocated query task. Given a query, when searching results using Web search engines, interacting with Chatbot4QR for evaluating the CQs, and judging the relevance of SO questions and Web search results, the participants should be based on the existing technical context specified in the query and/or their technical background. For example, for the query Q6 "*Better way to parse xml*", it has no specified programming language, the participants can perform the user studies with their preferred programming languages. For the query Q46 "*Choosing elements from python list based on probability*", it has a programming language Python. The participants should perform the user studies based on Python, but they can determine the other technical context, e.g., a Python library, based on their technical background.

## 5 Experiment Results

In this section, we answer the five research questions by conducting the corresponding user studies shown in Table 6.

### 5.1 RQ1: What are the proper settings of the parameters $n$ and $\eta$ in Chatbot4QR?

**Motivation**. In Chatbot4QR, $n$ and $\eta$ are two key parameters for generating CQs and recommending SO questions for queries. The settings of $n$ and $\eta$ will affect the quality of generated CQs and recommended questions. It is necessary to figure out the proper settings of the parameters.

**Approach**. We randomly selected ten queries from the 50 experimental queries and allocated the queries to the five participants in PG0. Then, we conducted the pilot **user study 1** shown in Fig. 5 as follows.

1. **CQ generation using different settings of $n$.** We generated different CQs for each query by setting $n$ from 5 to 50 with a step size 5.
2. **Usefulness evaluation of CQs.** We gathered the CQs generated using different values of $n$ for each query. The participants used the interactive interface of our Chatbot4QR prototype to evaluate the CQs. Before evaluation, we gave a tutorial using a video conference call with the participants to introduce the prototype with an example query outside the experimental query set. Then, the participants evaluated the CQs of each query by performing two tasks: (1) rate the usefulness of each CQ by five grades ranging from 0 to 4, where 0, 1, 2, 3, and 4 mean '*strongly useless*', '*useless*', '*neutral*', '*useful*', and '*strongly useful*', respectively; and (2) give feedback to the useful CQs. The **usefulness** of a CQ is judged by whether the CQ can help recognize any important information missed in a query for question retrieval.
3. **Sensitivity analysis of $n$.** For each setting of $n$, we counted the numbers of CQs with different usefulness and measured the ratio of useful CQs that are rated as 3 or 4 for each query. The usefulness of skipped CQs was deemed to 0; and we considered the usefulness of the CQs that were not prompted to the participants (due to the early termination of interaction) as unknown, because such CQs were not evaluated by the participants. Then, we determined a proper value of $n$ according to the results.
4. **Similarity adjustment using different settings of $\eta$.** Using the participants' feedback to the CQs generated with the proper $n$, we adjusted the initial semantic similarities of the 10,000 SO questions retrieved for each query (see Section 4.2) by setting $\eta$ from 0 to 1 with a step size 0.1.
5. **Relevance evaluation of SO questions.** We gathered the top ten SO questions obtained using different values of $\eta$ for each query. The participants evaluated the relevance of each question by five grades 0-4, where 0, 1, 2, 3, and 4 mean '*strongly irrelevant*', '*irrelevant*', '*neutral*', '*relevant*', and '*strongly relevant*', respectively. In the aforementioned video conference, we explained to the participants that the relevance of a SO question to a query should be judged by evaluating the degree of matching between the SO question and the query task

with the specified technical context (i.e., the technical terms appearing in the original query or given by the participants to the CQs).

6. **Sensitivity analysis of $\eta$.** For each setting of $\eta$, we measured the average performance of the top ten SO questions obtained for the ten queries using two metrics: Pre@k (Precision at k) [32] and NDCG@k (Normalized Discounted Cumulative Gain at k) [31], which are widely adopted in the IR community. Pre@k measures the percentage of relevant questions that are rated as 3 or 4 in the top-$k$ ranking list. NDCG@k considers the ranking and rating scores of relevant questions.

$$Pre@k = \frac{\#\text{ relevant questions in the top-}k}{k} \tag{3}$$

$$NDCG@k = \frac{1}{IDCG_k} \sum_{i=1}^{k} \frac{2^{rel_i} - 1}{log_2(1+i)} \tag{4}$$

where $rel_i$ is the relevance score of the question at the ranking position $i$; and $IDCG_k$ represents the maximum possible DCG score through position $k$ that can achieve for a query. Then, we determined a proper setting of $\eta$ according to the performance results.

**Results**. Figure 6 shows the numbers of three kinds of CQs generated for ten queries using different $n \in [5, 50]$, with respect to each of the five participants P1-P5 in PG0. "*Useful CQs*" are the CQs rated as 3 or 4. "*Useless & Neutral CQs*" are the CQs rated as 0, 1, or 2. "*Unknown CQs*" are the CQs with unknown usefulness. From the figure, we have the following findings:

- Under each setting of $n$, the total numbers of CQs generated for the five participants are different. For example, 15 (= 7+8) and 18 (= 2+16) CQs were generated for the participants P1 and P2, respectively, when $n$ = 5. This result is because that during the interaction, Chatbot4QR can dynamically generate subsequent CQs based on the participants' feedback to the initially generated CQs, as illustrated in Fig. 4. In particular, the participants have their own personalized technical background; and their feedback to CQs can be varied. Therefore, it leads to different numbers of CQs.
- There are notable differences among the five participants with respect to the numbers of the three kinds of CQs. For example, when $n$ = 5, only eight of the 15 CQs generated for P1 were evaluated as useful, while P2 evaluated 16 of the 18 generated CQs as useful. This result indicates that the participants had personalized judgement on the usefulness of CQs. Moreover, there are unknown CQs in the evaluation results of P2 and P5 when $n$ is a little large (e.g., $n$ = 25 for P2), meaning that some participants may only pay attention to a limited number of CQs during the interaction.
- For the five participants, at least 93.1% (= 27/29) of the useful CQs are generated by setting $n$ = 15. When $n$ is larger than 15, only one or two CQs are evaluated as useful by P2 and P5, while the number of useless, neutral, and unknown CQs increases. Therefore, we determine that $n$ = 15 is a good setting for Chatbot4QR.

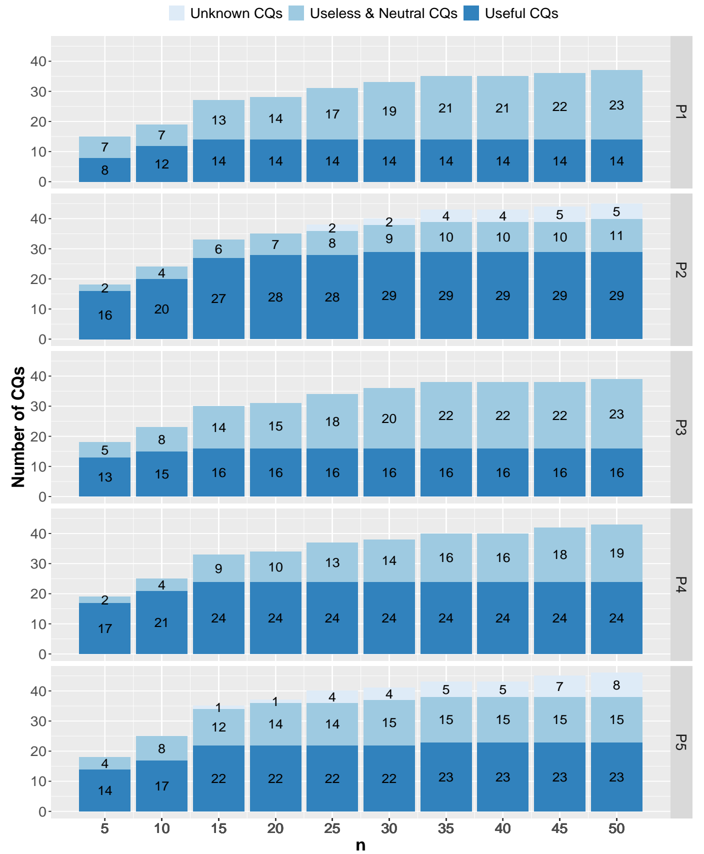


Fig. 6. The numbers of three kinds of CQs generated for ten queries using different settings of the parameter $n$ (i.e., ranging from 5 to 50). $n$ is the number of the initial top similar SO questions used for CQ generation. "*P1-P5*" are five participants.

*In Chatbot4QR, the parameter $n$, i.e., the number of the initial top similar SO questions used for CQ generation, is suggested to be set as 15.*

Table 7 presents the detailed evaluation results of each participant on the CQs generated for each query using $n$ = 15. "*#Initial CQs*" is the number of CQs that are initially generated by Chatbot4QR before interacting with the participants. "*#CQs*" is the total number of CQs generated after interacting with each participant. "*Ratio of Useful CQs*" is the ratio of useful CQs to the total CQs. We observe that the values of "*#CQs*" and "*Ratio of Useful CQs*" vary from the participants. For example, for the query Q5, the participant P3 got seven CQs (i.e., one CQ was dynamically generated), while the other participants got eight CQs (i.e., two CQs were dynamically generated). The ratios of useful CQs for P2, P4, and P5 are more than 0.75 and much higher than those for P1 and P3. These results show that our chatbot can generate personalized CQs based on the individual interaction with a participant; and the participants have personalized judgement on the usefulness of CQs.

Table 8 presents the average performance of the top ten SO questions retrieved using different $\eta \in [0, 1]$ by leveraging the participants' feedback to the CQs generated with $n$ = 15. From the table, we have the following findings:

- The performance achieved with a positive $\eta$ is much better than that achieved with $\eta$ = 0.0, indicating that

TABLE 7
Evaluation of the CQs generated using the initial top 15 similar SO questions retrieved for ten queries (i.e., setting the parameter $n$ = 15). "*P1-P5*" are five participants. "*#Initial CQs*" is the number of CQs that are initially by Chatbot4QR before interacting with the participants. "*#CQs*" is the number of CQs eventually generated by Chatbot4QR based on the participants' personalized feedback to CQs.

| Query No. | #Initial CQs | P1 | | P2 | | P3 | | P4 | | P5 | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | #CQs | Ratio of Useful CQs | #CQs | Ratio of Useful CQs | #CQs | Ratio of Useful CQs | #CQs | Ratio of Useful CQs | #CQs | Ratio of Useful CQs |
| 5 | 6 | 8 | 0.500 | 8 | 0.750 | 7 | 0.429 | 8 | 0.750 | 8 | 0.875 |
| 14 | 4 | 5 | 0.400 | 8 | 0.875 | 5 | 0.600 | 6 | 1.000 | 7 | 0.857 |
| 15 | 2 | 3 | 0.667 | 3 | 1.000 | 3 | 1.000 | 3 | 1.000 | 3 | 1.000 |
| 21 | 4 | 5 | 0.800 | 5 | 1.000 | 5 | 0.600 | 6 | 0.833 | 5 | 0.800 |
| 26 | 6 | 7 | 0.429 | 8 | 0.875 | 7 | 0.429 | 7 | 0.571 | 7 | 0.429 |
| 31 | 2 | 3 | 1.000 | 3 | 1.000 | 3 | 1.000 | 3 | 0.667 | 3 | 1.000 |
| 35 | 1 | 2 | 1.000 | 2 | 1.000 | 2 | 1.000 | 2 | 1.000 | 2 | 0.500 |
| 42 | 3 | 4 | 0.750 | 4 | 0.750 | 4 | 0.500 | 4 | 0.750 | 4 | 0.500 |
| 45 | 2 | 3 | 1.000 | 3 | 1.000 | 3 | 0.667 | 2 | 0.500 | 3 | 0.667 |
| 48 | 5 | 5 | 0.400 | 5 | 0.800 | 5 | 0.400 | 6 | 0.667 | 5 | 0.600 |

TABLE 8
The average performance of the top ten SO questions retrieved by Chatbot4QR for ten queries using different settings of the parameter $\eta$ (i.e., ranging from 0.0 to 1.0). $\eta$ is the weight coefficient of the participants' feedback to CQs in Eq. 1.

| $\eta$ | Pre@1 | Pre@5 | Pre@10 | NDCG@1 | NDCG@5 | NDCG@10 |
|---|---|---|---|---|---|---|
| 0.0 | 0.480 | 0.456 | 0.358 | 0.453 | 0.506 | 0.558 |
| 0.1 | 0.720 | 0.652 | 0.518 | 0.653 | 0.728 | 0.788 |
| 0.2 | 0.840 | **0.680** | **0.550** | 0.741 | **0.764** | **0.821** |
| 0.3 | **0.900** | 0.648 | 0.502 | **0.783** | 0.743 | 0.790 |
| 0.4 | **0.900** | 0.616 | 0.482 | **0.783** | 0.727 | 0.764 |
| 0.5 | 0.880 | 0.576 | 0.462 | 0.765 | 0.697 | 0.736 |
| 0.6 | 0.820 | 0.556 | 0.442 | 0.719 | 0.679 | 0.708 |
| 0.7 | 0.800 | 0.536 | 0.430 | 0.710 | 0.665 | 0.698 |
| 0.8 | 0.760 | 0.536 | 0.428 | 0.675 | 0.650 | 0.681 |
| 0.9 | 0.760 | 0.516 | 0.414 | 0.675 | 0.625 | 0.664 |
| 1.0 | 0.760 | 0.516 | 0.398 | 0.675 | 0.624 | 0.653 |

the participants' feedback to CQs can indeed help retrieve more relevant SO questions.

- As $\eta$ increases from 0.0 to 1.0, the Pre@k and NDCG@k values increase first until reach a peak; thereafter they decreases. This result can be explained by the fact that a query typically contains only a few keywords, a relatively large $\eta$ can overweight the user's technical feedback. Consequently, the recommended questions can match the user's technical requirements perfectly but are irrelevant to the programming problem.
- The optimal Pre@1 and NDCG@1 are achieved when $\eta$ = 0.3 or 0.4. When $k$ = 5 and 10, the optimal Pre@k and NDCG@k are achieved with $\eta$ = 0.2. Based on these results, there are two proper settings of $\eta$ depending on the user's desired number of recommended questions. If a user focuses on the top one question, it is suggested to set $\eta$ = 0.3 or 0.4, otherwise $\eta$ = 0.2 is suggested. Moreover, in terms of Pre@1 and NDCG@1, the performance achieved with $\eta$ = 0.2 is close to the optimal performance. Therefore, it is also a simple and good suggestion to set $\eta$ = 0.2, regardless of the value of $k$.

*In Chatbot4QR, for simplicity, the weight coefficient $\eta$ in Eq. 1 used for generating the recommended SO questions is suggested to be set as 0.2.*

### 5.2 RQ2: How effective can Chatbot4QR generate CQs?

**Motivation.** Our work is the first attempt to automatically generate CQs to interactively refine queries with the user involvement, in order to retrieve more relevant technical questions from Q&A sites. We want to evaluate the effectiveness of Chatbot4QR for CQ generation and verify whether the CQs can help users recognize missing technical details in queries.

**Approach.** We conducted a user study (i.e., the **user study 3** shown in Fig. 5) to evaluate the CQs generated by Chatbot4QR for the 50 experimental queries, under the setting of $n$ = 15. To the best of our knowledge, there is a similar work proposed by Rao et al. [17], named *EVPI*, which aims to generate CQs for asking good technical questions in Q&A sites. Unlike Chatbot4QR that can automatically generate CQs, *EVPI* extracts the existing CQs in the comment sections of the top ten similar questions retrieved using Lucene. Figure 7 shows two example CQs in the comment section of a SO question[5]. We implemented *EVPI* using the source code released at Github[6], and used *EVPI* to generate CQs for each experimental query.

5. https://stackoverflow.com/questions/22867636
6. https://github.com/raosudha89/ranking_clarification_questions

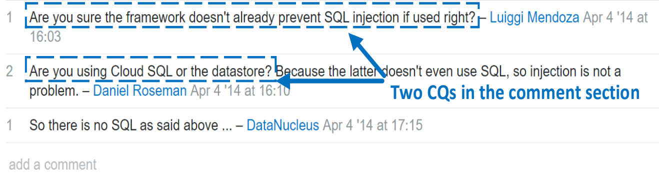


Fig. 7. Two CQs in the comment section of a SO question.

The 20 participants in PG1 and PG2 were required to evaluate the two kinds of CQs (one kind is generated by Chatbot4QR and the other kind is generated by *EVPI*) for their 25 allocated queries in QG1 and QG2, respectively. We modified the interactive interface of our Chatbot4QR prototype to run for the CQs generated by *EVPI*. More specifically, the prototype automatically prompted each query and the two kinds of CQs generated for the query in random order. The participants did not know which kind of CQs were generated by Chatbot4QR or *EVPI*. After completing the evaluation of CQs for a query, the participants needed to choose a preferred kind of CQs (i.e., the first or the second prompted kind). Before starting the evaluation, we launched a video conference to introduce the modified prototype to the participants with an example query. Then, the participants used the prototype to evaluate the two kinds of CQs for each allocated query by performing three tasks:

1. Rate the usefulness of each CQ by five grades 0-4, which are defined in Section 5.1.
2. Give feedback to the useful CQs.
3. Specify the preferred kind of CQs (when both Chatbot4QR and *EVPI* generated a set of CQs).

Note that the three tasks are not mandatory. The participants had the freedom to choose to perform any of the tasks. More specifically, the participants can skip a CQ if they think it is useless or feel difficult to answer. The participants can terminate the interaction with the chatbot early when they think that they have answered enough CQs for a query. If the participants have no preference for any of the two kinds of CQs, they can skip the Task 3. Since the participants may not know some technical terms asked in the CQs, they can search for unfamiliar technical terms (e.g., OpenCV and Keras) online during the interaction. Moreover, we asked the participants to manually record the amount of time spent on the interaction with Chatbot4QR for 25 allocated queries, as the participants can take a short break during the user study in case of personal work or fatigue. After the user study, we interviewed the participants to obtain their comments about the CQs produced by both approaches.

For each query, we counted the numbers of CQs generated by *EVPI* and *Chatbot4QR*, and measured the average ratio of useful CQs evaluated by the ten participants who were responsible for the query. We considered the usefulness of skipped CQs as 0 and excluded the CQs that were not displayed to the participants as the usefulness of such CQs was unknown. We also analyzed the participants' preferred kinds of CQs for the queries that have CQs generated by both approaches. We first identified two sets of participants for a query who preferred Chatbot4QR or *EVPI*, which are denoted as $P_{Chatbot4QR}$ and $P_{EVPI}$, responsively. Then, we defined the "**preference ratio**" of the ten participants for the query as $|P_{Chatbot4QR}|:|P_{EVPI}|$.

TABLE 9
Evaluation of the CQs generated by Chatbot4QR and *EVPI*. For a query, "*#Initial CQs*" is the number of CQs that are initially generated by Chatbot4QR; "*Avg. #CQs*" is the average number of CQs generated by Chatbot4QR after the interaction with ten participants; and "*#CQs*" is the number of CQs generated by *EVPI*.

| Query No. | CQs Generated by Chatbot4QR | | | CQs Generated by EVPI | |
|---|---|---|---|---|---|
| | #Initial CQs | Avg. #CQs | Avg. Ratio of Useful CQs | #CQs | Avg. Ratio of Useful CQs |
| 1 | 2 | 3 | **0.833** | 1 | 0.400 |
| 2 | 3 | 4 | **0.750** | 2 | 0.250 |
| 3 | 3 | 4.4 | **0.565** | 2 | 0.000 |
| 4 | 4 | 4.6 | **0.590** | 0 | – |
| 5 | 7 | 9 | **0.522** | 1 | 0.000 |
| 6 | 5 | 6 | **0.500** | 0 | – |
| 7 | 3 | 4 | 0.425 | 1 | 0.000 |
| 8 | 9 | 9.9 | 0.314 | 1 | 0.400 |
| 9 | 2 | 2.4 | **0.750** | 1 | 0.000 |
| 10 | 4 | 5.9 | **0.607** | 1 | 0.400 |
| 11 | 3 | 4.9 | **0.590** | 2 | 0.200 |
| 12 | 5 | 6.3 | 0.412 | 0 | – |
| 13 | 2 | 3 | **0.733** | 1 | 0.000 |
| 14 | 5 | 7.1 | **0.541** | 0 | – |
| 15 | 3 | 4 | **0.775** | 2 | 0.200 |
| 16 | 7 | 9 | 0.496 | 1 | 0.000 |
| 17 | 2 | 2.8 | **0.783** | 4 | 0.000 |
| 18 | 6 | 7.7 | 0.488 | 1 | 0.000 |
| 19 | 6 | 8.2 | 0.449 | 0 | – |
| 20 | 3 | 4 | **0.900** | 1 | 0.000 |
| 21 | 5 | 6.1 | **0.624** | 0 | – |
| 22 | 4 | 5 | **0.620** | 2 | 0.150 |
| 23 | 3 | 3 | **0.500** | 3 | 0.000 |
| 24 | 2 | 2.8 | **0.750** | 4 | 0.325 |
| 25 | 6 | 7.1 | **0.577** | 2 | 0.250 |
| 26 | 6 | 7 | 0.471 | 0 | – |
| 27 | 4 | 4.5 | **0.512** | 0 | – |
| 28 | 3 | 4 | **0.775** | 2 | 0.250 |
| 29 | 4 | 4 | **0.700** | 0 | – |
| 30 | 3 | 4.5 | **0.710** | 2 | 0.250 |
| 31 | 2 | 3 | **0.800** | 2 | 0.200 |
| 32 | 5 | 6 | **0.642** | 2 | 0.300 |
| 33 | 3 | 3.2 | **0.767** | 1 | **0.600** |
| 34 | 7 | 8.9 | 0.479 | 1 | 0.200 |
| 35 | 2 | 3.6 | **0.725** | 1 | **0.700** |
| 36 | 5 | 6.4 | **0.626** | 2 | 0.200 |
| 37 | 5 | 7.3 | **0.664** | 1 | 0.300 |
| 38 | 8 | 8.4 | 0.419 | 1 | 0.000 |
| 39 | 5 | 5 | **0.553** | 1 | 0.100 |
| 40 | 4 | 5 | 0.460 | 2 | 0.300 |
| 41 | 3 | 4 | **0.775** | 2 | 0.300 |
| 42 | 4 | 4.9 | **0.565** | 2 | 0.100 |
| 43 | 4 | 4.8 | **0.595** | 3 | 0.267 |
| 44 | 4 | 5 | 0.480 | 1 | **0.600** |
| 45 | 2 | 3 | **0.933** | 1 | 0.100 |
| 46 | 4 | 4 | **0.600** | 2 | 0.300 |
| 47 | 2 | 2.7 | 0.483 | 0 | – |
| 48 | 7 | 7.1 | 0.377 | 1 | **0.600** |
| 49 | 4 | 4.8 | **0.570** | 1 | 0.100 |
| 50 | 2 | 2 | **0.600** | 1 | 0.000 |
| **Avg.** | **4.1** | **5.1** | **0.608** | **1.3** | **0.167** |

Furthermore, according to the 20 types of SO tags shown in Table 1 and the three heuristic rules for CQ generation described in Section 3.2.2, Chatbot4QR can generate CQs that ask for 40 types of technical details, i.e., 20 types of SO tags and the versions. To examine whether the CQs that ask for some specific types of technical details would be more likely to be perceived as useful by users, we counted the numbers of CQs that ask for different types of technical details. We also measured the ratio of useful CQs that ask

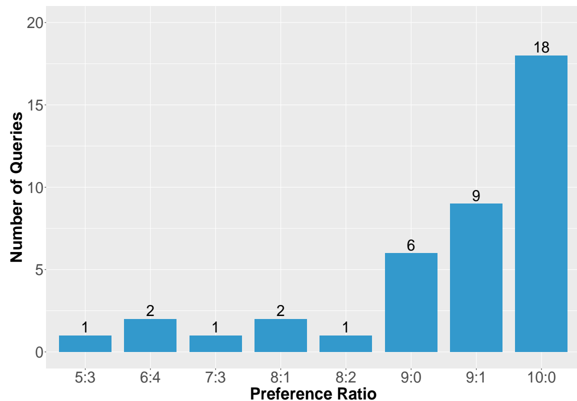


Fig. 8. Preference ratios of the CQs that are generated by Chatbot4QR and *EVPI* for 40 queries.

TABLE 10
Statistics on the evaluation of the CQs generated by Chatbot4QR and *EVPI*.

| Approach | #CQs Evaluated by the Participants | #Useful CQs Evaluated by the Participants |
|---|---|---|
| EVPI | 650 | 131 |
| Chatbot4QR | 2,565 | 1,479 |

for each type.

**Results.** Table 9 presents the numbers of CQs generated by Chatbot4QR and *EVPI*, as well as the average ratio of useful CQs for each query. For Chatbot4QR, we present the number of initially generated CQs and the average number of CQs (i.e., "*Avg. #CQs*") obtained by the ten participants in PG1 or PG2 after interaction, for each query. The bottom row shows the overall average results of both approaches on the 50 queries. From the table, we have the following findings:

- As for *EVPI*, it generated 1.3 CQs for a query on average and generated zero CQs for ten queries. The overall ratio of useful CQs for 50 queries is 16.7%, meaning that only a few CQs generated for a query were useful. Obviously, *EVPI* failed to generate any useful CQs for some vague queries. For example, the query Q6, i.e., "*Better way to parse xml*", is vague due to the missing of a specific programming language. However, no CQ was generated by *EVPI* for Q6.
- As for Chatbot4QR, on average for a query, it initially generated 4.1 CQs and finally generated 5.1 CQs after interacting with the participants. This result means that on average one CQ was dynamically generated for a query based on the participants' feedback. More specifically, 0-2.2 new CQs were generated for the 50 queries during the interaction. We observe that the number of CQs generated by Chatbot4QR is approximately four times the number of CQs generated by *EVPI*. Compared with *EVPI*, the effectiveness of Chatbot4QR depends on the number of increased useful CQs. If more than 16.7% of the increased CQs were useful, the effectiveness of Chatbot4QR would be better than that of *EVPI*. For each approach, we counted the number of times the generated CQs are evaluated by the participants and the number of times the CQs are evaluated as useful, as shown in Table 10. Among the 1,915 (=2,565-650) additional evaluations of the CQs generated by Chatbot4QR, 1,348 (i.e., 70.4%) are useful. Moreover, as listed in Table 9, the overall ratio of useful CQs that are generated by Chatbot4QR is 60.8% for the 50 queries. For 37 queries, the average ratios of useful CQs generated by Chatbot4QR are no less than 50%. In contrast, the average ratios of useful CQs generated by *EVPI* are no less than 50% for only four queries, i.e., Q33, Q35, Q44, and Q48. As demonstrated in the results, Chatbot4QR in CQ generation for a query is more effective than *EVPI*.

Figure 8 shows the numbers of queries with different preference ratios. There are 18 queries with the preference ratio '*10:0*', meaning that for these queries, all the ten participants preferred the CQs generated by Chatbot4QR. For the nine queries with '*5:3*', '*8:1*', and '*9:0*', one or two participants had no preference on the two kinds of CQs. We observe that most of the participants preferred the CQs generated by Chatbot4QR for the 40 queries (that have CQs generated by both approaches).

Two major comments about *EVPI* given by the participants are: (1) most of the generated CQs are too specific to a particular problem and often not useful for retrieving relevant questions, e.g., the CQ "*What exactly is a week number in this context, and what does 'date.weekday' have to do with it?*" generated for the query Q10; and (2) even some CQs might be useful but they are difficult to answer with a few words, e.g., the CQ "*What output did you get?*" generated for Q7. These issues can be explained by the objective of *EVPI* that it aims at generating CQs to help users refine technical questions, so that the questions can be easier to answer. Therefore, most of the CQs generated by *EVPI* may not be useful for question retrieval.

Table 11 presents the CQs generated by Chatbot4QR for the two queries Q18 and Q42. For the CQs related to each query, we present (1) the scores for ranking the CQs (see Section 3.2.2), (2) the orders of prompting the CQs to a participant (i.e., P16 for Q18 and P7 for Q42), and (3) the usefulness and feedback given by the participant. The two CQs without scores are dynamically generated based on the participants' feedback, e.g., P16's feedback '*windows*' to the third CQ of Q18. We observe that the ratios of useful CQs for Q18 and Q42 are 57.1% and 60.0%, respectively. We can use the final two kinds of feedback collected from all CQs of a query $q$ (i.e., the positive feedback $pfb(q)$ and negative feedback $nfb(q)$) to refine $q$. More specifically, $pfb(q)$ and $nfb(q)$ are used to adjust the initial semantic similarities of SO questions retrieved for $q$, as demonstrated in Eq. 1.

Table 12 presents the numbers of CQs and useful CQs generated by Chatbot4QR that ask for 30 types of technical details (including 18 technique types of SO tags and the versions of 12 technique types). The types with '*(v)*' are the versions of the corresponding technique types. For example, '*programming language (v)*' means the version of a programming language. The first row shows that there are 370 CQs that ask for programming languages; and 350 (i.e., 94.6%) of the CQs are evaluated as useful by the participants. We observe that the top five types of technical details asked by the maximum numbers of CQs are '*programming language (v)*', '*programming language*', '*library*', '*framework*',

TABLE 11
The CQs generated by Chatbot4QR for the two queries Q18 and Q42 in Table 4. For the CQs related to each query, we present the scores for ranking the CQs, the orders of the CQs prompted to a participant (i.e., P16 for Q18 and P7 for Q42), as well as the usefulness and feedback given by the participant. $pfb(q)$ and $nfb(q)$ are the positive feedback and negative feedback to all the CQs of a query $q$, respectively.

| Query No. | CQs Generated by Chatbot4QR for the Query | Score for CQ Ranking | Prompting Order | Usefulness | Feedback | Positive and Negative Feedback Used to Refine the Query |
|---|---|---|---|---|---|---|
| 18 | What programming language, e.g., javascript or python, does your problem refer to? | 1.000 | 1 | 0 | | pfb(Q18) = {node.js, windows 8, babel} |
| | If you are using a framework, e.g., node.js or mocha, please specify: | 0.604 | 2 | 3 | node.js | |
| | Could you provide an operating system, e.g., windows or linux? | 0.198 | 3 | 4 | windows | |
| | Can you specify the version of windows, e.g., 7 or 8? | – | 4 | 4 | 8 | |
| | Which library, e.g., package.json or babel, are you using? | 0.167 | 5 | 3 | babel | |
| | Are you using json? (y/n), or some other formats. | 0.135 | 6 | 0 | | |
| | Are you using terminal? (y/n), or some other classess. | 0.066 | 7 | 0 | | |
| 42 | What programming language, e.g., java or c#, does your problem refer to? | 1.000 | 1 | 3 | python | pfb(Q42) = {python 2.7, numpy}<br>nfb(Q42) = {.net, indexing} |
| | Can you specify the version of python, e.g., 3.x or 2.7? | – | 2 | 3 | 2.7 | |
| | Are you using .net? (y/n), or some other frameworks. | 0.033 | 3 | 1 | n | |
| | Are you using numpy? (y/n), or some other libraries. | 0.032 | 4 | 4 | numpy | |
| | Are you using indexing? (y/n), or some other techniques. | 0.032 | 5 | 2 | n | |

TABLE 12
The numbers of CQs and useful CQs generated for the 50 queries by Chatbot4QR that ask for different types of technical details, as well as the ratios of useful CQs. Each type with "*(v)*" means the version of the corresponding technique type.

| Type | #CQs that ask for the Type | #Useful CQs that ask for the Type | Ratio of Useful CQs that ask for the Type |
|---|---|---|---|
| programming language | **370** | 350 | **0.946** |
| programming language (v) | **494** | 422 | 0.854 |
| database | 20 | 15 | **0.750** |
| database (v) | 11 | 7 | 0.636 |
| operating system | **130** | 84 | **0.646** |
| operating system (v) | 61 | 37 | 0.607 |
| library | **329** | 179 | **0.544** |
| library (v) | 49 | 33 | 0.673 |
| technique | 129 | 48 | **0.372** |
| class | 98 | 35 | 0.357 |
| class (v) | 19 | 7 | 0.368 |
| non-PL language | 89 | 31 | 0.348 |
| non-PL language (v) | 47 | 29 | 0.617 |
| format | **130** | 37 | 0.285 |
| format (v) | 72 | 52 | 0.722 |
| model/algorithm | 19 | 5 | 0.263 |
| model/algorithm (v) | 22 | 21 | 0.955 |
| tool | 50 | 12 | 0.240 |
| tool (v) | 3 | 0 | 0.000 |
| framework | **239** | 55 | 0.230 |
| framework (v) | 22 | 14 | 0.636 |
| design pattern | 10 | 2 | 0.200 |
| environment | 59 | 10 | 0.169 |
| environment (v) | 2 | 0 | 0.000 |
| non-OS system | 70 | 11 | 0.157 |
| non-OS system (v) | 1 | 1 | 1.000 |
| platform | 10 | 1 | 0.100 |
| engine | 10 | 1 | 0.100 |
| server | 10 | 1 | 0.100 |
| browser | 20 | 1 | 0.050 |

and '*operating system*'. The top five types with the highest ratios of useful CQs are '*non-OS system (v)*', '*model/algorithm (v)*', '*programming language*', '*programming language (v)*', and '*database*'. Moreover, excluding the version types, the top five technique types with the highest ratios of useful CQs are '*programming language*', '*database*', '*operating system*', '*library*', and '*technique*'.

*On average, Chatbot4QR generates approximately five CQs for a query and 60.8% of the CQs are helpful for users to recognize missing technical details in the query. The CQs generated by Chatbot4QR are much better than the ones generated by the EVPI approach (as only 16.7% of the CQs generated by EVPI are helpful). Moreover, the CQs generated by Chatbot4QR that ask for some specific types of technical details are more likely to be perceived as useful by users. For the 20 types of SO tags shown in Table 1, the top five types with the highest ratios of useful CQs are 'programming language', 'database', 'operating system', 'library', and 'technique'.*

### 5.3 RQ3: Can Chatbot4QR retrieve more relevant SO questions than the state-of-the-art question retrieval and query expansion approaches?

**Motivation.** The ultimate goal of Chatbot4QR is to retrieve accurate SO questions for users based on their feedback to the CQs. Although it has been validated in RQ2 that most of the CQs generated by Chatbot4QR for a query are useful, it is necessary to check whether the refined queries (i.e., the participants' feedback to CQs) can improve the relevance of recommended questions.

**Approach.** We retrieved the top ten SO questions for the 50 experimental queries based on the participants' feedback to the CQs of each query, under the setting of $\eta$ = 0.2. Then, we conducted a user study (i.e., the **user study 5** shown in Fig. 5) to evaluate the SO questions. We compared Chatbot4QR with several existing question retrieval and query expansion approaches. More specifically, we summarized two state-of-the-art approaches used for question retrieval:

- *Lucene*: This is the Lucene search engine which retrieves SO questions similar to a query based on the Lucene index built for a question repository [8]. We implemented *Lucene* using its source code[7] released at Github.
- *Word Embedding (WE)*: This is the word embedding-based question retrieval approach widely used in recent work [5], [6]. We implemented *WE* using the source code[8] released by Huang et al. [5].

A rich body of research work improves the performance of IR systems by reformulating queries using relevant terms extracted from thesauruses or similar resources. We summarized three major query expansion approaches:

- *WordNet (WN)*: This approach expands a query with the synonyms of keywords in WordNet. We implemented the WordNet-based query expansion approach proposed by Lu et al. [12].
- *QECK*: This approach expands a query using the important keywords contained in the top similar SO question-and-answer pairs [8]. The importance of a keyword is measured by considering both the TF-IDF score and the scores of SO questions and answers. We implemented *QECK* according to the details presented in the paper.
- *Tag Recommendation (TR)*: There are a number of papers on recommending SO tags for a technical question [36],

7. https://github.com/apache/lucene-solr
8. https://github.com/tkdsheep/BIKER-ASE2018

[37], [38]. These papers are similar to Chatbot4QR to a certain extent as all of them focus on finding relevant SO tags for a target (a question or a query). We viewed the *TR* approaches as a specific kind of query expansion approaches, to check whether they can be used to recommend SO tags for queries. We implemented the neutral network-based *TR* approach proposed by Liu et al. [38] using the open-source code[9] and expanded a query with the top ten recommended SO tags.

We built nine baselines by combining the two retrieval approaches: *Lucene* and *WE*, and four query expansion approaches: *WN*, *QECK*, *TR*, and *IQR* (which refers to our interactive query refinement approach used in Chatbot4QR). The baselines are described as follows.

1. *Lucene*: This is the *Lucene* approach described above.
2. *WE*: This is the *WE* approach described above.
3. *Lucene+WN*: This approach uses *Lucene* to retrieve questions after expanding a query using *WN*.
4. *Lucene+QECK*: This approach uses *Lucene* to retrieve questions after expanding a query using *QECK*.
5. *Lucene+TR*: This approach uses *Lucene* to retrieve questions after expanding a query using *TR*.
6. *Lucene+IQR*: This approach uses *Lucene* to retrieve questions based on the query refined using *IQR*, i.e., the user's positive and negative feedback to CQs. More specifically, We first retrieved similar questions by applying the query and positive feedback to *Lucene*. Then, we removed the similar questions that contain any negative feedback.
7. *WE+WN*: This approach uses *WE* to retrieve questions after expanding a query using *WN*.
8. *WE+QECK*: This approach uses *WE* to retrieve questions after expanding a query using *QECK*.
9. *WE+TR*: This approach uses *WE* to retrieve questions after expanding a query using *TR*.

Note that Chatbot4QR can be simply viewed as a combination of *WE+IQR*. We applied the eight baselines except *Lucene+IQR* to the 50 queries and obtained eight lists of the top ten SO questions for each query. Since *IQR* is a personalized query refinement approach, we applied *Lucene+IQR* to retrieve the top ten questions based on each participant's feedback to the CQs of each query. Then, for each participant, we collected the different top ten questions retrieved for each query using Chatbot4QR and nine baselines. The participants evaluated the relevance of the questions by five grades 0-4, as defined in Section 5.1.

As the participants may probably have different preferences of techniques (e.g., the familiar programming languages shown in Table 5), they may get different SO questions retrieved by Chatbot4QR and *Lucene+IQR* for a query. Moreover, the participants may have their own judgement on the relevance of the questions. Therefore, we measured the overall Pre@k or NDCG@k performance of an approach $A$ as its average performance evaluated by the 20 participants. More specifically, given a specific Pre@k or NDCG@k metric $m$, for each participant $P$, we computed $m$ of each query according to $P$'s evaluation results of the SO questions retrieved by $A$. Then, we computed the average $m$ of the 25 queries allocated to $P$. Finally, we computed the overall $m$ of $A$, denoted as $m_A$, with respect to the average of the $m$ values of 20 participants. Based on the overall performance results, we measured the "**improvement degree**" of Chatbot4QR over each baseline $B$ in terms of a specific metric $m$ as $\frac{m_{Chatbot4QR} - m_B}{m_B}$.

Furthermore, we examined whether the improvement of Chatbot4QR (denoted as $C$) over a baseline $B$ is statistically significant. Considering that the participants may obtain different SO questions and have personalized benchmarks of relevant questions for a query, we defined a metric "**significant ratio**" to measure the statistical significance of the performance improvement of $C$ over $B$ as follows. For each participant, given a specific Pre@k or NDCG@k metric $m$, we built two samples for $C$ and $B$, respectively, by gathering the $m$ values of $C$ and $B$ on the 25 assigned queries. We used the Wilcoxon signed-rank test [39] to test the significance of $C$ over $B$ based on the two samples with three p-values {0.05, 0.01, 0.001}. For each p-value $p$, we identified the set of participants whose samples of $C$ are significantly better than those of $B$, which are denoted as $SigP_{m,p}(C, B)$. Then, the significant ratio of $C$ over $B$, given $m$ and $p$, is measured as

$$SigR_{m,p}(C, B) = \frac{|\ SigP_{m,p}(C, B)\ |}{\text{\# participants}}. \quad (5)$$

Finally, we chose the maximum significant ratio and the corresponding p-value.

As described in Section 5.2, the CQs generated by Chatbot4QR can ask for different types of technical details. In RQ2, we measured the ratios of useful CQs that ask for 30 types of technical details (see Table 12). We further measured the contributions of the participants' feedback to the CQs that ask for different types of technical details, in order to retrieve relevant questions. More specifically, for every feedback to a CQ of a query, we produced a list of the top ten SO questions by adjusting the initial semantic similarities of the 10,000 questions (see Section 4.2) using the single feedback. The participants evaluated the relevance of the questions that were not evaluated before for each allocated query. Then, for a query $q$, we computed the performance of a specific Pre@k and NDCG@k metric $m$ improved by each feedback $fb$, denoted as $Imp_m(q, fb)$, as follows.

- If $fb$ is a technique (e.g., a programming language) or a version given to an initially generated CQ, then $Imp_m(q, fb)$ is measured as $m_{Chatbot4QR}(q, fb) - m_{Initial}(q)$. $m_{Chatbot4QR}(q, fb)$ is the $m$ value of the top-$k$ questions retrieved for $q$ using Chatbot4QR by leveraging $fb$; and $m_{Initial}(q)$ is the $m$ value of the initial top-$k$ questions retrieved for $q$ using our two-phase method.
- If $fb$ is a version of a technique feedback $fb'$, then $Imp_m(q, fb)$ is measured as $m_{Chatbot4QR}(q, fb) - m_{Chatbot4QR}(q, fb')$.

Finally, we measured the average performance improvement achieved using the participants' feedback that belongs to a specific type of technical details. For each type, we also measured the average performance improvement achieved using the participants' feedback to all CQs of the queries that contain any feedback of the type.

**Results.** Table 13 presents the performance of the top ten SO questions retrieved using ten approaches. Table 14 presents

9. https://pan.baidu.com/s/1slujtU1

TABLE 13
Evaluation of the SO questions retrieved by ten approaches.

| Approach | Pre@1 | Pre@5 | Pre@10 | NDCG@1 | NDCG@5 | NDCG@10 |
|---|---|---|---|---|---|---|
| Lucene | 0.414 | 0.332 | 0.279 | 0.369 | 0.369 | 0.396 |
| Lucene+WN | 0.308 | 0.237 | 0.216 | 0.300 | 0.283 | 0.315 |
| Lucene+QECK | 0.278 | 0.190 | 0.156 | 0.251 | 0.245 | 0.260 |
| Lucene+TR | 0.250 | 0.203 | 0.169 | 0.243 | 0.246 | 0.265 |
| Lucene+IQR | 0.540 | 0.434 | 0.343 | 0.480 | 0.478 | 0.496 |
| WE | 0.530 | 0.416 | 0.348 | 0.484 | 0.473 | 0.500 |
| WE+WN | 0.300 | 0.236 | 0.188 | 0.285 | 0.281 | 0.299 |
| WE+QECK | 0.310 | 0.232 | 0.201 | 0.269 | 0.269 | 0.293 |
| WE+TR | 0.352 | 0.232 | 0.209 | 0.319 | 0.289 | 0.318 |
| Chatbot4QR | **0.838** | **0.670** | **0.548** | **0.765** | **0.731** | **0.760** |

TABLE 14
Improvement degrees and the maximum significant ratios of Chatbot4QR over nine baselines. "*ImpD*" is the improvement degree. "*(p, SigR)*" are the maximum significant ratio and the corresponding p-value.

| Baseline | Pre@1 | | Pre@5 | | NDCG@1 | | NDCG@5 | |
|---|---|---|---|---|---|---|---|---|
| | ImpD(%) | (p, SigR(%)) | ImpD(%) | (p, SigR(%)) | ImpD(%) | (p, SigR(%)) | ImpD(%) | (p, SigR(%)) |
| Lucene | **102.4** | **(0.05, 100.0)** | **102.1** | **(0.05, 100.0)** | **107.0** | **(0.01, 95.0)** | **97.8** | **(0.01, 100.0)** |
| Lucene+WN | 172.1 | (0.05, 100.0) | 182.9 | (0.01, 100.0) | 154.5 | (0.05, 100.0) | 158.5 | (0.01, 100.0) |
| Lucene+QECK | 201.4 | (0.01, 100.0) | 251.9 | (0.01, 100.0) | 205.2 | (0.05, 100.0) | 197.6 | (0.001, 100.0) |
| Lucene+TR | 235.2 | (0.01, 100.0) | 229.7 | (0.001, 100.0) | 214.0 | (0.01, 100.0) | 197.3 | (0.001, 100.0) |
| Lucene+IQR | **55.2** | **(0.05, 85.0)** | **54.2** | **(0.05, 95.0)** | **59.4** | **(0.05, 90.0)** | **52.7** | **(0.05, 100.0)** |
| WE | **58.1** | **(0.05, 70.0)** | **60.9** | **(0.05, 95.0)** | **57.8** | **(0.05, 80.0)** | **54.6** | **(0.01, 95.0)** |
| WE+WN | 179.3 | (0.05, 100.0) | 183.9 | (0.01, 100.0) | 168.4 | (0.01, 100.0) | 160.0 | (0.001, 100.0) |
| WE+QECK | 170.3 | (0.05, 100.0) | 189.3 | (0.01, 100.0) | 184.5 | (0.01, 100.0) | 171.9 | (0.001, 100.0) |
| WE+TR | 138.1 | (0.05, 100.0) | 189.3 | (0.001, 100.0) | 139.3 | (0.05, 100.0) | 152.8 | (0.001, 100.0) |

the improvement degrees and the maximum significant ratios of Chatbot4QR over the nine baselines. "*ImpD(%)*" is the improvement degree expressed as a percentage; and "*(p, SigR(%))*" are the maximum significant ratio expressed as a percentage and the corresponding p-value $p$. From the two tables, we have the following findings:

- Chatbot4QR achieves the best performance in terms of both Pre@k and NDCG@k. The result demonstrates that the queries refined by *IQR* (i.e., our interactive query refinement approach) can improve the quality of SO questions retrieved by *WE*.
- Chatbot4QR improves the two popular baselines *WE* and *Lucene* by at least 54.6% and 97.8%, respectively. Chatbot4QR significantly outperforms *Lucene* for all the participants in terms of Pre@1, Pre@5, and NDCG@5. Although Chatbot4QR does not significantly outperform *WE* for all the participants, the significant ratios are all higher than 70%. This result indicates that the improvement of Chatbot4QR over *WE* is significant for at least 14 of the 20 participants.
- *Lucene+IQR* improves *Lucene* by at least 22.91%, which further demonstrates the effectiveness of our *IQR* approach in helping users refine queries and retrieve more relevant questions using *Lucene*.
- *WE* outperforms *Lucene* by at least 24.48%. This is because that *WE* can retrieve semantically similar questions for queries, while *Lucene* cannot due to the lexical gaps issue.
- *WE* outperforms *WE+WN*, *WE+QECK*, and *WE+TR* by at least 50.57%. *Lucene* outperforms *Lucene+WN*, *Lucene+QECK*, and *Lucene+TR* by at least 22.93%. These results may indicate that the three automatic query expansion approaches (i.e., *WN*, *QECK*, and *TR*) are not suitable for reformulating queries to improve the performance of question retrieval.

Table 15 presents the average performance improvement of question retrieval achieved using the participants' feedback to the CQs that ask for 29 types of technical details. For each type, "*# Cases*" is the number of times a participant's feedback of the type is used for question retrieval; "*Avg. Imp*" is the average Pre@k or NDCG@k improvement achieved using the participants' feedback of the type; and "*Avg. Imp by All*" is the average Pre@k or NDCG@k improvement achieved using the participants' feedback to all CQs of the queries that contain any feedback of the type. The type '*tool (v)*' in Table 12 has no improvement result in Table 15, since none of the three CQs that ask for the type of technical details are evaluated as useful and thus there is no feedback of the type used for question retrieval. We find that the feedback of some types has positive improvement while the feedback of other types has no or negative improvement. For example, in terms of Pre@1, the improvement of the three types '*programming language*', '*database (v)*', and '*operating system*' are 0.318, 0.000, and -0.207, respectively. We also find that the improvement achieved using the feedback of a type can be different in terms of different metrics. For example, the feedback of '*operation system*' has a negative impact on the Pre@1 and NDCG@1 performance, however, it has positive improvement in terms of Pre@5 and NDCG@5. The version types have very low improvement. One of the possible reasons could be that many SO questions do not explicitly specify the versions of the involved techniques, es-

TABLE 15
The average performance improvement of SO question retrieval achieved using the participants' feedback to the CQs generated by Chatbot4QR that ask for different types of technical details. Each type with "*(v)*" means the version of the corresponding technique type. For each type, "*#Cases*" is the number of times a participant's feedback of the type is used to adjust the initial semantic similarities of SO questions of a query; "*Avg. Imp*" is the average Pre@k or NDCG@k improvement achieved using the participants' feedback of the type; "*Avg. Imp by All*" is the average Pre@k or NDCG@k improvement achieved using the participants' feedback to all CQs of the queries that contain any feedback of the type.

| Type | #Cases | Pre@1 | | Pre@5 | | NDCG@1 | | NDCG@5 | |
|---|---|---|---|---|---|---|---|---|---|
| | | Avg. Imp | Avg. Imp by All | Avg. Imp | Avg. Imp by All | Avg. Imp | Avg. Imp by All | Avg. Imp | Avg. Imp by All |
| programming language | 355 | **0.318** | 0.363 | **0.243** | 0.299 | **0.263** | 0.333 | **0.248** | 0.307 |
| programming language (v) | 454 | 0.075 | 0.326 | 0.040 | 0.264 | 0.065 | 0.285 | 0.039 | 0.265 |
| database | 15 | **0.333** | 0.333 | **0.173** | 0.240 | **0.236** | 0.271 | **0.179** | 0.268 |
| database (v) | 8 | 0.000 | 0.375 | 0.000 | 0.325 | 0.000 | 0.317 | 0.000 | 0.334 |
| operating system | 92 | -0.207 | 0.217 | 0.039 | 0.241 | -0.120 | 0.266 | 0.003 | 0.264 |
| operating system (v) | 45 | 0.000 | 0.200 | 0.018 | 0.276 | 0.000 | 0.276 | 0.016 | 0.290 |
| library | 183 | **0.279** | 0.421 | **0.118** | 0.280 | **0.224** | 0.381 | **0.133** | 0.293 |
| library (v) | 37 | -0.081 | 0.324 | -0.022 | 0.254 | -0.011 | 0.275 | -0.008 | 0.242 |
| technique | 56 | 0.054 | 0.464 | -0.004 | 0.289 | 0.076 | 0.397 | 0.019 | 0.297 |
| class | 35 | -0.057 | 0.343 | -0.063 | 0.217 | -0.094 | 0.306 | -0.071 | 0.240 |
| class (v) | 13 | 0.000 | 0.077 | 0.000 | 0.431 | 0.000 | 0.021 | -0.014 | 0.262 |
| non-PL language | 33 | -0.364 | 0.273 | 0.012 | 0.406 | -0.322 | 0.227 | -0.063 | 0.307 |
| non-PL language (v) | 20 | 0.000 | 0.450 | 0.010 | 0.350 | 0.000 | 0.378 | -0.028 | 0.259 |
| format | 40 | -0.175 | 0.175 | -0.035 | 0.200 | -0.108 | 0.140 | -0.036 | 0.202 |
| format (v) | 61 | 0.000 | 0.197 | 0.013 | 0.216 | 0.000 | 0.165 | 0.004 | 0.186 |
| model/algorithm | 5 | -0.400 | 0.200 | -0.120 | -0.000 | -0.107 | 0.333 | -0.109 | 0.057 |
| model/algorithm (v) | 21 | 0.000 | -0.143 | 0.010 | 0.124 | 0.000 | -0.113 | 0.008 | 0.099 |
| tool | 15 | 0.000 | 0.200 | 0.040 | 0.213 | 0.036 | 0.196 | 0.047 | 0.233 |
| framework | 63 | 0.127 | 0.286 | 0.029 | 0.235 | 0.116 | 0.312 | 0.052 | 0.245 |
| framework (v) | 17 | -0.176 | 0.235 | -0.047 | 0.141 | -0.024 | 0.267 | -0.017 | 0.212 |
| design pattern | 3 | 0.000 | 0.333 | 0.067 | 0.467 | 0.000 | 0.311 | 0.066 | 0.521 |
| environment | 14 | -0.143 | 0.143 | -0.014 | 0.171 | -0.062 | 0.248 | -0.024 | 0.199 |
| environment (v) | 1 | 0.000 | 0.000 | 0.000 | 0.200 | 0.000 | 0.533 | 0.000 | 0.276 |
| non-OS system | 14 | 0.000 | 0.500 | 0.086 | 0.400 | -0.038 | 0.479 | 0.046 | 0.403 |
| non-OS system (v) | 1 | 0.000 | 1.000 | 0.000 | 0.400 | 0.000 | 0.933 | 0.000 | 0.401 |
| platform | 3 | 0.000 | 1.000 | 0.000 | 0.200 | 0.000 | 1.000 | 0.000 | 0.333 |
| engine | 1 | 0.000 | 0.000 | 0.000 | 0.200 | 0.000 | 0.000 | -0.054 | 0.025 |
| server | 1 | 0.000 | 0.000 | 0.000 | 0.400 | 0.000 | 0.000 | 0.000 | 0.203 |
| browser | 2 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.083 |

pecially in the question title and tags. To effectively leverage the feedback of versions, there needs a method for inferring the versions of techniques from the content (e.g., code snippets) of questions. Moreover, the values of "*Avg. Imp by All*" are generally higher than those of "*Avg. Imp*", indicating that better performance is achieved by integrating the feedback to all CQs of a query. The top three types with the maximum performance improvement are the same, i.e., '*programming language*', '*database*', and '*library*', in terms of all metrics.

*Compared with the nine baselines that involve two question retrieval approaches and four query expansion approaches, Chatbot4QR retrieves more relevant SO questions for queries. The improvement degree of Chatbot4QR over the word embedding-based question retrieval approach (WE) is at least 54.6%. Furthermore, the improvement of Chatbot4QR over WE is statistically significant for more than 70% of the participants. Moreover, the participants' feedback to the CQs that ask for different types of technical details has different contributions to question retrieval. The top three types with the maximum contributions are 'programming language', 'database', and 'library'.*

### 5.4 RQ4: How efficient is Chatbot4QR?

**Motivation.** In Chatbot4QR, several resources need to be built offline, including the Lucene index of SO questions, two language models, and the categorization and version-frequency information of SO tags. Although the offline processing takes a substantial amount of time, the built resources are reusable. We are interested in finding out the response time that Chatbot4QR can respond to a user once the user submits a query. If the response time is too long, our approach may not be acceptable even if it is effective in generating useful CQs and retrieving relevant SO questions. Therefore, it is essential to examine whether Chatbot4QR is efficient for practical uses.

**Approach.** We recorded the amount of time that Chatbot4QR, *WE*, and *Lucene* spent on the offline processing of SO data and online question retrieval during our experiments. After the **user study 3**, we asked the participants to report their time spent on interacting with our chatbot. We did not consider the time costs of the other seven baselines because *Lucene+IQR* is based on our *IQR* and the performance of other baselines is too low (see Table 13).

**Results.** Table 16 presents the time costs of the three approaches. From the table, we have the following findings:

- As for the offline processing, the processing time of Chatbot4QR is 91.15 hours, which is much higher than those of *Lucene* and *WE*. This is because that the offline processing of Chatbot4QR contains three main parts: (1) the semi-automatic categorization of SO tags (74 hours); (2) the building of the Lucene index of SO

TABLE 16
Time costs of three approaches.

| Approach | Offline Processing | Online Question Retrieval |
|---|---|---|
| Lucene | 8.52h | 0.02s |
| WE | 7.38h | 49.96s |
| Chatbot4QR | 91.15h | **Response: 1.30s** |
| | | Interaction: ≈ 42s |
| | | Recommendation: 0.02s |

questions and two language models (8.52+7.38 = 15.9 hours); and (3) the tag identification from SO questions (1.25 hours). Since the resources are reusable and can be incrementally updated (as explained in Section 3.1.5), the relatively high time cost of the offline processing of Chatbot4QR may not be a problem for practical uses.

- As for the online question retrieval for a query, the processing time of Chatbot4QR contains three parts (as shown in Table 16): (1) *Response* is the amount of time required to respond to a participant (1.30 seconds), including the two-phase question retrieval and CQ generation; (2) *Interaction* is the amount of time that a participant spent on the interaction with our chatbot (about 42 seconds); and (3) *Recommendation* is the amount of time required to adjust the similarities of 10,000 SO questions and produce the top ten recommended questions (0.02 seconds). The response time is 1.30 seconds, meaning that Chatbot4QR can responsively start interacting with the user after receiving a query. After the interaction, the question recommendation list can be produced within 0.02 seconds. These results demonstrate the efficiency of Chatbot4QR.
- The time spent by *WE* on question retrieval is 49.96 seconds per query, which is high because *WE* measures the semantic similarities between a query and the 1,880,269 SO questions in our repository. In contrast, the two-phase question retrieval approach used in Chatbot4QR is scalable. The reason is that the first phase uses *Lucene*, which is efficient to handle a large-scale repository, as shown in Table 16; and by fixing the parameter $N$ to a relatively large value (e.g., 10,000 in this work), the time cost of the second phase is stable.

It is worth to mention that the average time spent on the interaction with our chatbot for a query is 42 seconds. For a few queries, some participants took 2-3 minutes because they needed to search for unfamiliar technical terms asked in the CQs online. **As confirmed by the participants, the amount of time spent on the interaction is practically acceptable since the feedback to CQs can contribute to more relevant SO questions and reduce the time required for the manual examination of undesirable questions**. Although the quality of retrieved questions relies on the user's feedback to CQs, Chatbot4QR does not require the user to answer every CQ. The amount of the interaction time depends on (1) the user's programming experience and (2) whether the user wants to search for unfamiliar technical terms online, in order to provide more precise feedback to CQs and obtain more relevant questions.

*Chatbot4QR takes approximately 1.30 seconds to respond to a user after the user submits a query and 0.02 seconds to produce the SO question recommendation list after interacting with the user, indicating that Chatbot4QR is efficient for practical uses.*

### 5.5 RQ5: Can Chatbot4QR help obtain better results than using Web search engines alone?

**Motivation.** In practice, developers often use the SO search engine and general-purpose search engines (e.g., Google) to look for desired information [13], [40]. To further validate the effectiveness of Chatbot4QR, we investigate whether Chatbot4QR can help users obtain better results than using Web search engines (including the SO search engine, Google, etc.) alone. Here, a result refers to a SO question or any other resources returned by Web search engines, e.g., a blog or a tutorial.

**Approach.** We conducted four user studies (i.e., the **user studies 2, 4, 5**, and **6** shown in Fig. 5) for answering RQ5. Before the interaction with Chatbot4QR, we asked the 20 participants in PG1 and PG2 to obtain the top ten results using Web search engines of their choices for each allocated query. The participants can modify a query according to the returned results until they are satisfied with the results (excluding the SO question whose title is the same as the original query) listed in a webpage. For each query, we asked the participants to record the final query and the top ten results in the returned webpage. After interacting with Chatbot4QR, the participants can obtain new results for a query using Web search engines by reformulating the query with interesting technical terms in their feedback to the CQs. Take the query Q18 shown in Table 11 as an example, the participant P16 can reformulate Q18 by adding some technical terms, e.g., '*node.js*', in his/her positive feedback. Then, the participants evaluated the relevance of the two kinds of Web search results. Finally, the participants chose the preferred/best results for each allocated query from the three kinds of results: the top ten SO questions retrieved by Chatbot4QR and the two top ten Web search results. After the user studies, we interviewed the participants to get their opinions on the value of Chatbot4QR.

For each participant, we measured the performance of the following three top ten results for each allocated query:

- *WS*: the top ten results obtained using Web search engines before the interaction with Chatbot4QR.
- *WS+IQR*: the top ten results obtained using Web search engines after the interaction with Chatbot4QR.
- *Best*: the best results chosen by the participant.

We viewed *WS*, *WS+IQR*, and *Best* as three retrieval approaches. For a specific Pre@k or NDCG@k metric, we measured the overall performance of each approach as the average performance evaluated by the 20 participants. Moreover, we measured the improvement degrees and the maximum significant ratios of *Best* over *WS* and *WS+IQR*. The detailed measurement process can refer to Section 5.3.

**Results.** Table 17 presents the overall performance of *WS*, *WS+IQR*, and *Best*, as well as the improvement degrees and the maximum significant ratios of *Best* over *WS* and *WS+IQR*. From the table, we have the following findings:

- *Best* outperforms *WS* and *WS+IQR* in terms of both Pre@k and NDCG@k with an improvement of at least

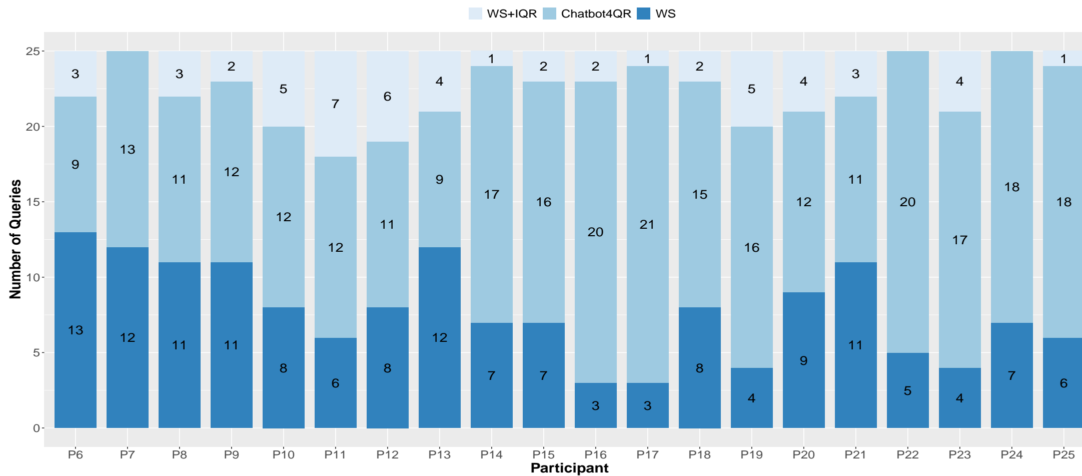


Fig. 9. The numbers of queries that achieve the best results using *WS*, *WS+IQR*, and Chatbot4QR by 20 participants.

TABLE 17
Evaluation of the results obtained using Web search engines before/after interacting with Chatbot4QR. "*ImpD*" is the improvement degree. "*(p, SigR)*" are the maximum significant ratio and the corresponding p-value.

| | Pre@1 | Pre@5 | Pre@10 | NDCG@1 | NDCG@5 | NDCG@10 |
|---|---|---|---|---|---|---|
| WS | 0.634 | 0.483 | 0.401 | 0.532 | 0.500 | 0.502 |
| WS+IQR | 0.664 | 0.524 | 0.433 | 0.555 | 0.528 | 0.531 |
| Best | **0.900** | **0.725** | **0.585** | **0.798** | **0.746** | **0.749** |
| ImpD(%) of Best over WS | 22.4 | 29.4 | 26.9 | 27.5 | 26.9 | 29.8 |
| (p, SigR (%)) of Best over WS | (0.05, 80.0) | (0.05, 100.0) | (0.05, 90.0) | (0.05, 90.0) | (0.01, 100.0) | (0.01, 100.0) |
| ImpD(%) of Best over WS+IQR | 16.9 | 19.3 | 17.3 | 22.3 | 20.0 | 22.5 |
| (p, SigR (%)) of Best over WS+IQR | (0.05, 70.0) | (0.05, 95.0) | (0.05, 85.0) | (0.05, 85.0) | (0.01, 100.0) | (0.05, 100.0) |

22.4% and 16.9%, respectively. The significant ratios of *Best* over *WS* are all higher than 80%, indicating that the improvement of *Best* over *WS* is statistically significant for at least 16 of the 20 participants. This result shows that more desired results are obtained by the participants after interacting with Chatbot4QR than directly using Web search engines.

- *WS+IQR* is slightly better than *WS*. This means that for some queries, the participants obtained better results using Web search engines again by reformulating the queries using information that they learned from the interaction with our chatbot.

We counted the numbers of queries that achieve the best results by *WS*, *WS+IQR*, and Chatbot4QR for each participant, as shown in Fig. 9. From the figure, we have the following findings:

- For 12-22 of the 25 (i.e., 48%-88%) assigned queries, the participants preferred the results obtained by Chatbot4QR or *WS+IQR*. For 16 participants (except P6, P8, P13, and P21), Chatbot4QR achieves the best results for the largest number of queries. Moreover, there are 1-7 queries whose best results are obtained by *WS+IQR* for 17 participants (except P7, P22, and P24). For example, for the query Q22, the participant P3 reformulated it by adding the feedback '*mysql*' given to the CQ "*I want to know whether you are using a database, e.g., mysql or mongodb. Can you provide it?*", which contributes to the best results retrieved by Google. All these results are consistent with the overall performance shown in Table 17, which further demonstrate that for a considerable number of queries, Chatbot4QR helps the participants obtain better results than using Web search engines alone.
- For each of the two participant groups PG1 (=P6-P15) and PG2 (=P16-P25), there are notable differences among the participants with respect to the numbers of queries whose best results are obtained by *WS*, *WS+IQR*, and Chatbot4QR. By interviewing the participants, we found that the differences are mainly caused by the participants' different preferences of techniques and programming experience. For example, for the query Q36, the participants P23 and P24 preferred Java while P25 preferred Python. Before using Chatbot4QR, P24 reformulated the query by adding '*jsoup*' [41] (a Java HTML parser) while P23 simply added '*java*'. Consequently, they obtained different results for Q36.
- For all the 20 participants, *WS* achieves the best results for 3-13 queries. We found that most of those queries are relatively simple and have specified technical terms, e.g., Q1 and Q7. The result shows the good performance of Web search engines when the query is clearly specified. **Although Chatbot4QR cannot achieve the best results for some queries, all the participants expressed their willingness to use our chatbot as a complement to Web search engines.**

Moreover, we examined the query reformulation records

TABLE 18
The numbers of participants who reformulated the 50 queries.

| Query Nos. | #Participants Who Reformulated the Queries |
|---|---|
| 2, 26, 45 | 8 |
| 13, 20, 41, 48 | 7 |
| 1, 21, 22, 25, 32, 37, 39, 46 | 6 |
| 3, 8, 12, 18, 19, 27, 28, 31, 36 | 5 |
| 5, 10, 15, 17, 29, 33, 42, 50 | 4 |
| 6, 14, 34, 35, 38, 40, 49 | 3 |
| 7, 11, 16, 24, 30, 44 | 2 |
| 23, 43 | 1 |
| 4, 9, 47 | 0 |

TABLE 19
Technical terms used to reformulate ten queries. Each number in a parenthesis is the frequency of the technical term used to reformulate the corresponding query.

| Query No. | Technical Terms Used to Reformulate the Query |
|---|---|
| 2 | numpy(3), python(2), matplotlib(2), python 3.x(1) |
| 26 | java(3), collections(1), c(1), python 3(1), linux(1) |
| 45 | pandas(7), python(1) |
| 13 | python(2), numpy(2), java(1), c#(1), python 3.x(1) |
| 48 | regex(4), python 3.x(1), jquery(1), django(1) |
| 7 | java(2) |
| 11 | java(2) |
| 16 | xml(1), html(1) |
| 23 | python 3(1) |
| 43 | neural-network(1) |

of the participants by leveraging the final queries that they used for obtaining the results of *WS* and *WS+IQR*. Table 18 presents the queries according to the number of participants who had reformulated them. We observe that 24 (=3+4+8+9) queries were reformulated by 5-8 participants, while 11 (=3+2+6) queries were reformulated by 0-2 participants. We further analyzed the participants' feedback to CQs used in their reformulated queries. Table 19 lists the statistics of the technical terms added to five frequently reformulated queries and five less frequently reformulated queries. The number in a parenthesis indicates the frequency of the corresponding technical term used to reformulate a query. As shown in Table 19, the queries reformulated by more participants often involve multiple techniques. For example, the technical terms used to reformulate the query Q26 include three programming languages {'*java*', '*c*', '*python 3*'}, one operating system {'*linux*'}, and one library {'*collections*'}.

*For 12-22 of the 25 (i.e., 48%-88%) assigned queries, the participants preferred the results obtained by Chatbot4QR or using Web search engines with the queries reformulated after interacting with Chatbot4QR. This demonstrates that Chatbot4QR can help obtain better results than using Web search engines alone. During the interview with the participants, all the participants expressed their willingness to use Chatbot4QR as a complement to Web search engines.*

## 6 Discussion

### 6.1 Why Chatbot4QR can help users retrieve better SO questions and Web search results?

Tables 13 and 14 show that Chatbot4QR significantly outperfoms the two popular retrieval approaches: *WE* and *Lucene*, as well as their variants combined with three query expansion approaches: *WN*, *QECK*, and *TR*. Moreover, *WE* is better than *Lucene* because *WE* can bridge the lexical gaps between SO questions and queries while *Lucene* cannot. The variants perform worse than *WE* and *Lucene* due to the fact that *WN*, *QECK*, and *TR* may introduce noise terms and decrease the quality of retrieved SO questions. As an example, for the query Q9, i.e., "*how to encrypt data using AES in Java*", the terms expanded by *QECK* are: {*ruby*, *iv*, *php*, *openssl*, *algorithm*, *i.e.fast*, *disk*, *byte*, *decrypt*}. Since Q9 has a programming language '*java*', the two terms '*php*' and '*ruby*' may probably be unexpected by users.

Based on the above analysis, Chatbot4QR uses *WE* as the question retrieval model. However, the performance of *WE* is limited by the quality of queries. When a query is vague, e.g., missing important technical details, *WE* cannot retrieve accurate questions. As users may have varied technical background, Chatbot4QR uses an interactive approach to assisting users in refining queries by asking CQs that are generated according to the missing technical details in a query. The user's feedback to CQs can accurately represent their technical requirements on the queries and contribute to retrieving relevant SO questions.

Table 17 and Fig. 9 show that Chatbot4QR helps the participants obtain much better results than using Web search engines alone for at least 48% of their allocated queries. The SO questions retrieved by Chatbot4QR were chosen as the best results by 16 of the 20 participants for the largest proportion of queries. For some queries, the participants obtained the best results using Web search engines by reformulating the queries with their feedback to CQs. These results demonstrate that Chatbot4QR can (1) retrieve desired SO questions for users after helping them refine the queries and (2) help users better understand their queries and obtain better results using Web search engines.

### 6.2 Why Not Use A Constant Chatbot?

Chatbot4QR is designed to generate different CQs for queries based on the existing technical details mentioned in a query and an initial set of similar SO questions retrieved for the query. *Is this design necessary? Can we use a constant chatbot that always asks several fixed CQs for queries?* To answer these questions, we implemented a constant chatbot, denoted as *ConstantBot*, which focuses on asking for four types of technical details given a query, namely the programming language, framework, and the versions of the two technique types. More specifically, *ConstantBot* first asks a CQ "*What programming language does your problem refer to?*". If a user provides a programming language, e.g., Java, then *ConstantBot* further asks for the version of Java using a CQ "*Can you specify the version of java?*". Then, *ConstantBot* asks for a framework using "*If you are using a framework, please specify:*", as well as the version of a possible framework given by the user.

TABLE 20
The average numbers of CQs and the average ratios of useful CQs that are generated by Chatbot4QR and *ConstantBot* for the 50 queries.

| Approach | Avg. #CQs | Avg. Ratio of Useful CQs |
|---|---|---|
| ConstantBot | 2.7 | 0.446 |
| Chatbot4QR | 5.1 | 0.608 |

TABLE 21
Evaluation of the SO questions retrieved by Chatbot4QR and *ConstantBot*; and the improvement degrees and the maximum significant ratios of Chatbot4QR over *ConstantBot*.

| | Pre@1 | Pre@5 | NDCG@1 | NDCG@5 |
|---|---|---|---|---|
| ConstantBot | 0.766 | 0.596 | 0.687 | 0.661 |
| Chatbot4QR | 0.838 | 0.670 | 0.765 | 0.731 |
| ImpD(%) | 9.4% | 12.3% | 11.3% | 10.5% |
| (p, SigR(%)) | (0.05, 20.0%) | (0.05, 55.0%) | (0.05, 30.0%) | (0.05, 35.0%) |

We asked the 20 participants in PG1 and PG2 to evaluate the CQs asked by *ConstantBot* for each allocated query. Similar to the **user study 3** conducted in Section 5.2, the participants rated each CQ by five grades 0-4 (as defined in Section 5.1) and gave feedback to the useful CQs. After the evaluation, we asked the participants to provide some comments about *ConstantBot*. Then, we retrieved the top ten similar SO questions using Eq. 1 by leveraging each participant's feedback to a query. The participants evaluated the relevance of the questions that were not evaluated before by five grades 0-4, as defined in Section 5.1.

Table 20 presents the average number of CQs and the average ratio of useful CQs that are asked by *ConstantBot* for the 50 queries. Table 21 presents the Pre@k and NDCG@k performance of the retrieved SO questions, as well as the improvement degrees and the maximum significant ratios of Chatbot4QR over *ConstantBot*. From Table 20, we find that on average *ConstantBot* asked 2.7 CQs for a query; and the ratio of useful CQs is 44.6%, which is much lower than that of Chatbot4QR (i.e., 60.8%). By analyzing the evaluation results of the 50 queries, there are 11 queries (i.e., Q4, Q9, Q17, Q23, Q27, Q29, Q32, Q33, Q39, Q46, and Q50) that have no useful CQ as evaluated by the participants. The 11 queries contain a specific programming language, e.g., Java in Q4; and the participants are not interested in looking for a framework. Two major comments about *ConstantBot* given by the participants are: (1) *ConstantBot* still asks for a programming language when a query already has a programming language; and (2) unlike Chatbot4QR, *ConstantBot* cannot help recognize some technical details that are useful but missed in a query, e.g., databases and libraries. From Table 21, we find that in terms of Pre@k and NDCG@k (k = 1 and 5) metrics, Chatbot4QR improves *ConstantBot* by 9.4%-12.3%; and the improvement is statistically significant for 20%-55% participants. Based on the analysis results, we can conclude that it is not appropriate to use a constant chatbot for the interactive query refinement and question retrieval.

### 6.3 Learning Effect from Interacting with Chatbot4QR

In Section 5.5, considering that the participants can learn to recognize some missing technical details in queries from the interaction with Chatbot4QR, we first asked the participants to search results for queries before interacting with Chatbot4QR. This can avoid the impact of the participants' learning effect on their Web search results using the original queries.

It is worth mentioning that the learning effect is good for users in practice. After interacting with Chatbot4QR for several times, users, especially the novices, can learn to formulate high-quality queries with necessary technical details for retrieving questions from SO or other resources from general-purpose Web search engines (e.g., Google). Because of the learning effect, users can ask better questions in Q&A sites by describing their problems with a clear technical context, which can lead to better answers. Moreover, there are too many techniques (e.g., libraries) available on the Web; and it is difficult for users, even for experienced developers, to know every possible technique. Chatbot4QR may help users, including both novices and experienced developers, discover unknown or better techniques for some programming tasks.

### 6.4 Application Scenarios of Chatbot4QR

Chatbot4QR can be applied in the following two scenarios:

- Chatbot4QR can be implemented as a browser plugin. When a user inputs a query to Web search engines, the plugin detects the missing technical details in the query. If there are missing technical details, the plugin informs the user that the query has a quality issue. Then, the user can choose to interact with our chatbot. After the interaction, our chatbot recommends the top ten similar SO questions. Moreover, the user can use their feedback to CQs to reformulate the query for Web search.
- In the literature, many technical tasks, such as answer summarization [6] and API recommendation [5] rely on the quality of the top similar SO questions retrieved for queries. Chatbot4QR could be used to improve the performance of question retrieval, which will contribute to better results of the tasks.

### 6.5 Participants' Comments about Chatbot4QR

In the experiments, we encouraged the participants to provide comments about Chatbot4QR. We summarize several major positive and negative aspects of the comments.

- **Positive Comments**

PC1. *The chatbot is good! It can really help me figure out some important technical details missed in the queries. And, the final retrieved results are more satisfactory.*

PC2. *Using the generated CQs to refine queries is more straightforward and systematic than manually picking up relevant information scents in the search results.*

PC3. *The chatbot is flexible and fast, and most of the asked CQs are really closely related to a query.*

PC4. *Although it may ask some unfamiliar techniques for me, it still helps me get a better understanding of the query as well as some other possibly useful libraries. I'd like to try it later.*

- **Negative Comments**

NC1. *Some CQs are unnecessary because the asked information can be inferred from some keywords in the query. For*

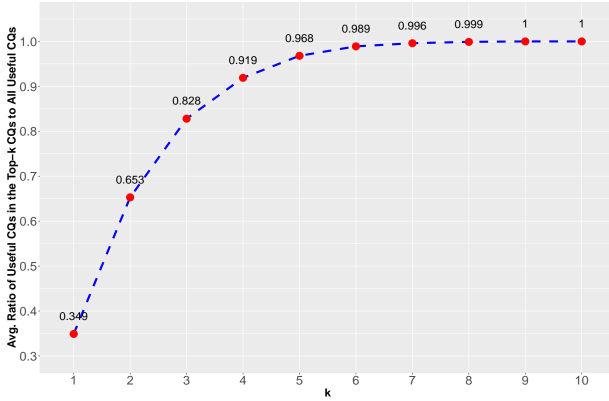


Fig. 10. The average ratios of useful CQs that are contained in the top-$k$ CQs prompted to the participants to all useful CQs for the 50 queries.

*example, the CQ "What programming language, e.g., java or c#, does your problem refer to?" asked for the query Q35 "Using LINQ to extract ints from a list of strings" is useless because 'LINQ' is based on C#.*

NC2. *There are a bit too many CQs for some queries. Although the chatbot allows me to skip and terminate, I suggest that you can limit the number of CQs for a query, e.g., five.*

According to the comments, Chatbot4QR can assist the participants in refining queries and retrieving more desired results (PC1 and PC2). Additionally, Chatbot4QR can help the participants better understand the queries and discover some possibly useful techniques (PC4). The efficiency of Chatbot4QR is also acceptable (PC3). However, there still remain some issues. For example, Chatbot4QR cannot filter unnecessary CQs based on the existing information in queries (NC1). To solve this issue, we need to mine the relationships among techniques, e.g., what frameworks and libraries are related to a specific programming language. Moreover, the participants suggest us to limit the number of CQs asked for a query (NC2).

To validate the suggestion in NC2, we generated a list of the CQs evaluated by a participant for a query. The CQ list was ranked by the orders of the CQs prompted to the participant during the interaction with Chatbot4QR. We counted the number of useful CQs in the top-$k$ CQs of the list, and measured the ratio of useful CQs in the top-$k$ to all useful CQs that are evaluated by the participant for the query. We set $k$ from 1 to 10 (the maximum number of CQs in all CQ lists). For each $k$, we measured the average ratio of useful CQs in all CQ lists, as shown in Fig. 10. On average, 96.8% of the useful CQs of a query are contained in the top five CQs prompted to a participant, indicating that it is suitable to limit the number of CQs asked for a query as 5.

### 6.6 Error Analysis of Chatbot4QR

Although it has been validated that Chatbot4QR can effectively generate useful CQs and recommend relevant SO questions for queries, we find two error scenarios of Chatbot4QR from the participants' evaluation results as follows.

1. As reported in the comment NC1, Chatbot4QR may generate wrong CQs for a query. Despite the wrong CQ asked for the query Q35 in NC1, the CQ "*What programming language, e.g., javascript or python, does your problem refer to?*" generated for the query Q18 (see Table 11) is also useless since the technical term '*NPM*' in Q18 is highly related to JavaScript. During the interaction with a participant, Chatbot4QR cannot dynamically filter unsuitable CQs or technical terms appearing in CQs based on the participant's feedback. For example, for the query Q42 shown in Table 11, after the participant P7 answered the CQ "*What programming language, e.g., java or c#, does your problem refer to?*" with Python, the subsequent CQ "*Are you using .net? (y/n), or some other frameworks.*" became unsuitable as '*.net*' is a C# framework. The CQ should be revised by replacing '*.net*' with a Python framework appearing in the initial top-$n$ similar questions retrieved for Q42. If there is no such a Python framework, the CQ can be removed. Through our analysis, the errors are caused by the fact that Chatbot4QR currently has no knowledge about the relationships between techniques, e.g., NPM is related to JavaScript and .net is related to C#. In the future, we plan to mine knowledge about the relationships among SO tags and integrate the knowledge to Chatbot4QR.
2. Chatbot4QR may produce worse question recommendation lists than the two-phase method for some queries. For example, for the query Q5 "*How to insert multiple rows into database using hibernate?*", the participant P17 provided three kinds of positive feedback, i.e., {'*java 8*', '*mysql*', '*sql*'}, to the CQs. However, the final top ten SO questions refined by incorporating the feedback are worse than the initial top ten questions. Table 22 presents the initial and the final top five questions retrieved for Q5, as well as the relevance of the questions evaluated by P17. By analyzing the results, the performance of the final top five questions is decreased as some questions (e.g., the question '23200729') are irrelevant to the query task, but they match all the feedback, and therefore the rankings of such questions are over-promoted. To correct such errors, our future work will aim to optimize the weights of different types of technical feedback in Eq. 1 according to their contributions to the question retrieval (see Table 15).

### 6.7 Threats to Validity

**Threats to internal validity** relate to two aspects in this work: (1) the errors in the implementation of Chatbot4QR and the baseline approaches and (2) the participants' bias during the experiments.

As for the aspect (1), we carefully checked the implementation code of our Chatbot4QR prototype. Considering that there could be noises in the tag assignments of SO questions, which may affect the CQ generation of Chatbot4QR, we built the question repository by requiring that each question has an accepted answer and a positive score. Moreover, we ensured that the experimental queries and their duplicates were not included in the repository. Although our experimental queries were built from the titles of SO questions, it may not be a serious problem as it is a common experimental setup used in previous work [5], [6], [8], [20], [32], [33]. For the four baselines *EVPI*, *Lucene*, *WE*, and *TR*, we directly used the open-source code. For the other two baselines *WN* and *QECK*, we carefully re-implemented them according to

TABLE 22
Two kinds of the top five SO questions retrieved for the query Q5; and the relevance of questions evaluated by the participant P17. "*Initial*" represents the top five questions retrieved using our two-phase method. "*Final*" represents the top five questions retrieved using Chatbot4QR by leveraging P17's feedback to the CQs of Q5.

| Result Type | The Top Five SO Questions | | | Relevance |
|---|---|---|---|---|
| | Question ID | Question Title | Question Tags | |
| Initial | 20045940 | Inserting multiple rows in database | java, database | 4 |
| | 47244614 | Inserting Data in Multiple Tables in Hibernate | java, hibernate, jpa | 3 |
| | 22553920 | Insert into two tables in two different database | java, spring, hibernate, jpa | 2 |
| | 39383049 | How to insert data to multiple table at once in hibernate using java | java, mysql, hibernate | 3 |
| | 22472292 | How to insert new items with Hibernate? | java, mysql, hibernate | 2 |
| Final | 25485086 | how to insert new row in hibernate framework? | java, mysql, sql, hibernate | 3 |
| | 23200729 | Records in DB are not one by one. Hibernate | java, mysql, sql, hibernate | 1 |
| | 39383049 | How to insert data to multiple table at once in hibernate using java | java, mysql, hibernate | 3 |
| | 31583737 | hibernate: how to select all rows in a table | java, mysql, sql, hibernate, postgresql | 2 |
| | 22472292 | How to insert new items with Hibernate? | java, mysql, hibernate | 2 |

the details presented in the papers [8], [12]. Therefore, there is little threat to the implementation of the approaches.

As for the aspect (2), we recruited the participants who are interested in our work and have 2-11 years of programming experience. We adopted several strategies to mitigate the participants' bias in the steps that require manual efforts. For the categorization of SO tags, we used two iterations of a card sorting approach. Each iteration step was independently conducted by the first two co-authors of the paper; then they worked together with an invited postdoc to discuss the disagreements to obtain the final results. We asked the participants to search results for the queries using Web search engines before interacting with Chatbot4QR, in order to avoid the participants transferring the knowledge learned from our chatbot to enhance the original queries when they use Web search engines. Before evaluating CQs in the user studies 1 and 3, we launched a video conference with the participants to introduce our Chatbot4QR prototype, to ensure that they understood how to use the prototype for evaluation. In the video conference of the user study 1, we also explained the relevance judgement of SO questions to a query with a technical context. Moreover, at the beginning of our user studies, we explained to the participants about how to perform the user studies based on the existing technical details in queries and/or their technical background. It is possible that the participants may have difficulties in building the technical context for some queries as they may not be interested in the problems. In the future, we plan to develop Chatbot4QR as a plugin and deploy the plugin in companies, such as Hengtian, to validate whether Chatbot4QR can help developers retrieve better SO questions or other resources for technical problems.

**Threats to external validity** relate to the generalizability of experiment results. To alleviate this threat, we built a large-scale repository of 1.88 million SO questions. To conduct our user studies, we recruited 25 participants. Considering that the user studies require significant manual efforts, we built 50 experimental queries. The number of participants and the number of experimental queries are close to the existing user studies in the previous work [30], [32], [33], [34], [35]. The 25 participants have different years of programming experience and diverse familiar programming languages, as shown in Table 5. The 50 experimental queries have diversity in the involved techniques, the complexity of problems, and the quality of expression (i.e., whether there are specified techniques or not), as explained in Section 4.2. The diversity of participants and queries can help improve the generalizability of our experiment results. In the future, we plan to further reduce this threat by extending the user studies with more participants and queries.

**Threats to construct validity** relate to the suitability of evaluation metrics. To reduce this threat, we used two popular metrics: Pre@k and NDCG@k, which are widely used to evaluate the ranking results in the fields of IR and software engineering [5], [6], [31], [32], [33], [42].

## 7 Related Work

**Question Retrieval in SO.** Question retrieval is a key step for many knowledge search tasks in SO. A number of work retrieves similar SO questions for queries by leveraging the Lucene search engine [8] or word embedding techniques [3], [5], [6], [11]. For example, Nie et al. [8] proposed a code search approach by expanding queries with important keywords extracted from relevant SO question-and-answer pairs. Lucene is used for indexing and retrieving SO question-and-answer pairs. Xu et al. [6] proposed an approach named *Answerbot* to generating a summarized answer for a query by extracting important sentences from the answers of similar SO questions. It retrieves similar SO questions using a word embedding-based approach. Huang et al. [5] proposed an API recommendation approach named *BIKER*. A word embedding-based approach is also used for retrieving SO questions similar to a query. The recommended APIs are extracted from the answers of the top ten similar SO questions. The Lucene search engine is efficient but cannot handle the lexical gaps between SO questions and queries. Recently, the word embedding-based approach is widely used to bridge the lexical gaps and can achieve better performance. However, the existing work on question retrieval rarely considers an important issue in practice that the query can be inaccurately specified, which will lead to undesireable questions.

We propose a novel question retrieval approach which improves the word embedding-based approach in two main aspects: (1) a two-phase question retrieval approach is used to improve the efficiency by reducing the search space using Lucene before applying the word embedding-based approach; and (2) a chatbot is designed to interactively help users refine queries by asking several CQs related to the

missing technical details in a query. The refined queries can contribute to retrieving more relevant SO questions.

**Tag Recommendation in SO.** SO encourages users to attach several (no more than five) tags to a question, which can help organize the tremendous amount of questions and facilitate the question retrieval [43]. However, the large set of more than 50 thousand SO tags imposes a huge burden for users to select a few appropriate tags for a question. Much attention has been paid to recommending relevant tags for SO questions [36], [37], [38], [43]. For example, Xia et al. [36] proposed an approach called *TagCombine* to finding relevant tags by composing three ranking components. Wang et al. [37] proposed a tag recommendation system by using the labeled Latent Dirichlet Allocation (LDA) modeling technique [44]. They analyzed the historical tag assignments and users of SO questions and the original tags provided by users. Zhou et al. [38] proposed a neural network approach to recommending tags, which leverages both textual descriptions and tags of SO questions.

Different from the tag recommendation (TR) work, our work focuses on determining missing technical details in a query based on the tags of similar SO questions. As evaluated in Section 5.3, the existing TR approaches may not be suitable for determining relevant tags for queries because of two main reasons. First, unlike a SO question that has a rich description (including the title, original tags, and body), a query typically consists of a few keywords, which makes it difficult to find relevant tags precisely. Second, even for the same query, different users may have personalized preferences of tags considering their different technical background (e.g., the preferred programming languages) or programming context (e.g., the platform the software is developed for). To address these challenging issues, given a query, we use a chatbot to interact with the user by asking several CQs with a candidate set of relevant tags extracted from the top-$n$ similar SO questions, allowing the user to tell what tags they want.

**Query Reformulation.** The quality of queries has an great impact on the performance of IR systems. However, it is not an easy task to formulate a good query, which largely depends on the user's experience and their knowledge about the IR system [45]. A lot of work has been proposed to automatically reformulate queries by expanding them with relevant terms extracted from lexical databases (e.g., WordNet) or similar resources [8], [12], [15], [46], [47]. For example, Lu et al. [12] proposed to expand a query with the synonyms in WordNet for code search. Nie et al. [8] also proposed a code search approach by expanding queries with important keywords extracted from relevant SO question-and-answer pairs. A major limitation of automatic query expansion approaches is that there can be unexpected terms added to the query without user involvement, which will affect the quality of results.

To overcome the limitation, several work has recently been proposed to interactively help users refine queries [31], [48], [49]. For example, Zou et al. [48] proposed a personalized Web service recommendation approach, which can assist users in refining their requirements. The approach is based on a process knowledge base built from the available online resources. Guo et al. [49] proposed an interactive image search approach which uses a reinforcement learning model to capture the user's feedback on their desired image. The approach relies on the predefined feature set of images. It has been demonstrated that these interactive query refinement approaches can help find desired results for users. In contrast to these work, we propose an interactive query refinement approach to assisting users in clarifying the missing technical details in queries, in order to improve the performance of question retrieval from technical Q&A sites. For this purpose, we build two technical knowledge bases, i.e., the categorization and multiple version-frequency information of SO tags.

## 8 Conclusion and Future Work

Question retrieval plays an important role in acquiring knowledge from technical Q&A sites, e.g., SO. The existing search engines provided in the Q&A sites and the state-of-the-art question retrieval approaches are insufficient to retrieve desired questions for users when the query is inaccurately specified. In this paper, we propose a chatbot, named Chatbot4QR, to interactively help users refine their queries for question retrieval. Chatbot4QR can accurately detect missing technical details in a query and interacts with the user by asking several CQs. The user's feedback to CQs is used to retrieve more relevant SO questions. The evaluation results of six user studies demonstrate the effectiveness and efficiency of Chatbot4QR.

To the best of our knowledge, it is the first work on the interactive query refinement for technical question retrieval. However, the current Chatbot4QR is still in infancy with some limited capability. In the current stage, Chatbot4QR focuses on helping users clarify 20 major types of techniques (see Table 1) and the versions of the techniques missed in a query. In the future, we will improve Chatbot4QR in two main directions: (1) we plan to use the possible solutions discussed in Section 6.6; and (2) we will mine knowledge on the differences (e.g., the frequency of use and performance) between the similar techniques (e.g., HashMap is more efficient than Hashtable [6]), so that Chatbot4QR could suggest better techniques when users intend to search for a less frequently used technique or an obsolete technique (e.g., Hashtable). Moreover, we plan to implement Chatbot4QR as a browser plugin to assist users in searching results for technical problems. When a user inputs a query to a Web search engine (e.g., the SO search engine or Google), the plugin can notify the user if there are missing technical details in the query. The user can interact with our chatbot to obtain the top ten similar SO questions and get insights for reformulating the query to search on the Web.


## Acknowledgment

This research was partially supported by the National Key R&D Program of China (No.2019YFB1600700), NSFC Program (No. 61972339), the Australian Research Council's Discovery Early Career Researcher Award (DECRA) (DE200100021), Ministry of Education, Singapore, under its Academic Research Fund Tier 2 (Award No.: MOE2019-T2-1-193), Natural Sciences and Engineering Research Council of Canada (NSERC), and Alibaba-Zhejiang University Joint Institute of Frontier Technologies.

# References


[1] C. Chen, X. Chen, J. Sun, Z. Xing, and G. Li, "Data-driven proactive policy assurance of post quality in community q&a sites," *Proceedings of the ACM on Human-Computer Interaction*, vol. 2, no. CSCW, p. 33, 2018.

[2] D. Ye, Z. Xing, and N. Kapre, "The structure and dynamics of knowledge network in domain-specific q&a sites: a case study of stack overflow," *Empirical Software Engineering*, vol. 22, pp. 375–406, 2017.

[3] B. Xu, Z. Xing, X. Xia, D. Lo, and S. Li, "Domain-specific cross-language relevant question retrieval," *Empirical Software Engineering*, vol. 23, pp. 1084–1122, 2018.

[4] E. C. Campos, L. B. de Souza, and M. d. A. Maia, "Searching crowd knowledge to recommend solutions for api usage tasks," *Journal of Software: Evolution and Process*, vol. 28, no. 10, pp. 863–892, 2016.

[5] Q. Huang, X. Xia, Z. Xing, D. Lo, and X. Wang, "Api method recommendation without worrying about the task-api knowledge gap," in *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*. ACM, 2018, pp. 293–304.

[6] B. Xu, Z. Xing, X. Xia, and D. Lo, "Answerbot: Automated generation of answer summary to developers' technical questions," in *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2017, pp. 706–716.

[7] H. Yin, Z. Sun, Y. Sun, and W. Jiao, "A question-driven source code recommendation service based on stack overflow," in *2019 IEEE World Congress on Services (SERVICES)*, vol. 2642. IEEE, 2019, pp. 358–359.

[8] L. Nie, H. Jiang, Z. Ren, Z. Sun, and X. Li, "Query expansion based on crowd knowledge for code search," *IEEE Transactions on Services Computing*, vol. 9, no. 5, pp. 771–783, 2016.

[9] S. Haiduc, G. Bavota, A. Marcus, R. Oliveto, A. De Lucia, and T. Menzies, "Automatic query reformulations for text retrieval in software engineering," in *Proceedings of the 2013 International Conference on Software Engineering*. IEEE Press, 2013, pp. 842–851.

[10] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Advances in neural information processing systems*, 2013, pp. 3111–3119.

[11] T. Nandi, C. Biemann, S. M. Yimam, D. Gupta, S. Kohail, A. Ekbal, and P. Bhattacharyya, "Iit-uhh at semeval-2017 task 3: Exploring multiple features for community question answering and implicit dialogue identification," in *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*, 2017, pp. 90–97.

[12] M. Lu, X. Sun, S. Wang, D. Lo, and Y. Duan, "Query expansion via wordnet for effective code search," in *2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*. IEEE, 2015, pp. 545–549.

[13] M. M. Rahman, J. Barson, S. Paul, J. Kayani, F. A. Lois, S. F. Quezada, C. Parnin, K. T. Stolee, and B. Ray, "Evaluating how developers use general-purpose web-search for code retrieval," in *Proceedings of the 15th International Conference on Mining Software Repositories*. ACM, 2018, pp. 465–475.

[14] G. A. Miller, *WordNet: An electronic lexical database*. MIT press, 1998.

[15] F. Pérez, J. Font, L. Arcega, and C. Cetina, "Collaborative feature location in models through automatic query expansion," *Automated Software Engineering*, vol. 26, no. 1, pp. 161–202, 2019.

[16] E. Linstead, S. Bajracharya, T. Ngo, P. Rigor, C. Lopes, and P. Baldi, "Sourcerer: mining and searching internet-scale software repositories," *Data Mining and Knowledge Discovery*, vol. 18, no. 2, pp. 300–336, 2009.

[17] S. Rao and H. Daumé III, "Learning to ask good questions: Ranking clarification questions using neural expected value of perfect information," in *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2018, pp. 2737–2746.

[18] "Google search," *https://www.google.com*.

[19] "Chatbot4qr release," *https://tinyurl.com/y6dgwwy5*, 2019.

[20] X. Ye, H. Shen, X. Ma, R. Bunescu, and C. Liu, "From word embeddings to document similarities for improved information retrieval in software engineering," in *Proceedings of the 38th international conference on software engineering*. ACM, 2016, pp. 404–415.

[21] "Tagwiki," *https://stackoverflow.com/tags*, 2019.

[22] S. Bird, E. Klein, and E. Loper, *Natural language processing with Python: analyzing text with the natural language toolkit*. O'Reilly Media, Inc., 2009.

[23] R. Rehurek and P. Sojka, "Software framework for topic modelling with large corpora," in *In Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*. Citeseer, 2010.

[24] 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 *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*. IEEE, 2016, pp. 90–101.

[25] C. Chen, S. Gao, and Z. Xing, "Mining analogical libraries in q&a discussions–incorporating relational and categorical knowledge into word embedding," in *2016 IEEE 23rd international conference on software analysis, evolution, and reengineering (SANER)*. IEEE, 2016, pp. 338–348.

[26] M. Nassif, C. Treude, and M. Robillard, "Automatically categorizing software technologies," *IEEE Transactions on Software Engineering*, 2018.

[27] Q. Huang, X. Xia, D. Lo, and G. C. Murphy, "Automating intention mining," *IEEE Transactions on Software Engineering*, 2018.

[28] J. L. Fleiss, "Measuring nominal scale agreement among many raters," *Psychological bulletin*, vol. 76, no. 5, p. 378, 1971.

[29] "Models of the information seeking process."

[30] H. Niu, I. Keivanloo, and Y. Zou, "Learning to rank code examples for code search engines," *Empirical Software Engineering*, vol. 22, no. 1, pp. 259–291, 2017.

[31] N. Zhang, J. Wang, Y. Ma, K. He, Z. Li, and X. F. Liu, "Web service discovery based on goal-oriented query expansion," *Journal of Systems and Software*, vol. 142, pp. 73–91, 2018.

[32] X. Gu, H. Zhang, and S. Kim, "Deep code search," in *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)*. IEEE, 2018, pp. 933–944.

[33] X. Li, Z. Wang, Q. Wang, S. Yan, T. Xie, and H. Mei, "Relationship-aware code search for javascript frameworks," in *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*. ACM, 2016, pp. 690–701.

[34] J. Zhang, H. Jiang, Z. Ren, T. Zhang, and Z. Huang, "Enriching api documentation with code samples and usage scenarios from crowd knowledge," *IEEE Transactions on Software Engineering*, 2019.

[35] A. Alami, M. L. Cohn, and A. Wasowski, "Why does code review work for open source software communities?" in *Proceedings of the 41st International Conference on Software Engineering*. IEEE Press, 2019, pp. 1073–1083.

[36] X. Xia, D. Lo, X. Wang, and B. Zhou, "Tag recommendation in software information sites," in *2013 10th Working Conference on Mining Software Repositories (MSR)*. IEEE, 2013, pp. 287–296.

[37] S. Wang, D. Lo, B. Vasilescu, and A. Serebrenik, "Entagrec++: An enhanced tag recommendation system for software information sites," *Empirical Software Engineering*, vol. 23, pp. 800–832, 2018.

[38] J. Liu, P. Zhou, Z. Yang, X. Liu, and J. Grundy, "Fasttagrec: fast tag recommendation for software information sites," *Automated Software Engineering*, vol. 25, no. 4, pp. 675–701, 2018.

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

[40] S. E. Sim, M. Umarji, S. Ratanotayanon, and C. V. Lopes, "How well do search engines support code retrieval on the web?" *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 21, no. 1, p. 4, 2011.

[41] "Jsoup," *https://jsoup.org*, 2019.

[42] N. Zhang, J. Wang, K. He, Z. Li, and Y. Huang, "Mining and clustering service goals for restful service discovery," *Knowledge and Information Systems*, vol. 58, no. 3, pp. 669–700, 2019.

[43] P. Zhou, J. Liu, Z. Yang, and G. Zhou, "Scalable tag recommendation for software information sites," in *2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 2017, pp. 272–282.

[44] G. Boudaer and J. Loeckx, "Enriching topic modelling with users' histories for improving tag prediction in q&a systems," in *Proceedings of the 25th International Conference Companion on World Wide Web*. International World Wide Web Conferences Steering Committee, 2016, pp. 669–672.

[45] J. A. Rodriguez Perez and J. M. Jose, "Predicting query performance in microblog retrieval," in *Proceedings of the 37th international ACM SIGIR conference on Research & development in information retrieval*. ACM, 2014, pp. 1183–1186.

[46] C. Lucchese, F. M. Nardini, R. Perego, R. Trani, and R. Venturini, "Efficient and effective query expansion for web search," in *Pro-*

*ceedings of the 27th ACM International Conference on Information and Knowledge Management*. ACM, 2018, pp. 1551–1554.

[47] M. M. Rahman and C. Roy, "Effective reformulation of query for code search using crowdsourced knowledge and extra-large data analytics," in *2018 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 2018, pp. 473–484.

[48] P. K. Venkatesh, S. Wang, Y. Zou, and J. W. Ng, "A personalized assistant framework for service recommendation," in *2017 IEEE International Conference on Services Computing (SCC)*. IEEE, 2017, pp. 92–99.

[49] X. Guo, H. Wu, Y. Cheng, S. Rennie, G. Tesauro, and R. Feris, "Dialog-based interactive image retrieval," in *Advances in Neural Information Processing Systems*, 2018, pp. 678–688.