Automated Query Reformulation for Efficient Search based on Query Logs From Stack Overflow

Kaibo Cao[†], Chunyang Chen[‡*], Sebastian Baltes[§], Christoph Treude[§], Xiang Chen[¶*]
[†]*Software Institute*, *Nanjing University*, China
[‡]*Faculty of Information Technology*, *Monash University*, Australia
[§]*School of Computer Science*, *University of Adelaide*, Australia
[¶]*School of Information Science and Technology*, *Nantong University*, China
imkbcao@gmail.com, chunyang.chen@monash.edu, {sebastian.baltes, christoph.treude}@adelaide.edu.au, xchencs@ntu.edu.cn

***Abstract*—As a popular Q&A site for programming, Stack Overflow is a treasure for developers. However, the amount of questions and answers on Stack Overflow make it difficult for developers to efficiently locate the information they are looking for. There are two gaps leading to poor search results: the gap between the user's intention and the textual query, and the semantic gap between the query and the post content. Therefore, developers have to constantly reformulate their queries by correcting misspelled words, adding limitations to certain programming languages or platforms, etc. As query reformulation is tedious for developers, especially for novices, we propose an automated software-specific query reformulation approach based on deep learning. With query logs provided by Stack Overflow, we construct a large-scale query reformulation corpus, including the original queries and corresponding reformulated ones. Our approach trains a Transformer model that can automatically generate candidate reformulated queries when given the user's original query. The evaluation results show that our approach outperforms five state-of-the-art baselines, and achieves a 5.6% to 33.5% boost in terms of $ExactMatch$ and a 4.8% to 14.4% boost in terms of $GLEU$.**

***Index Terms*—Stack Overflow, Data Mining, Query Reformulation, Deep Learning, Query Logs**

I. Introduction

Stack Overflow is the most popular question and answer (Q&A) site for programming-related knowledge sharing and acquisition. Over the past decade, Stack Overflow has accumulated a large amount of user-generated content, making it a valuable repository of software engineering knowledge. When developers encounter a specific programming question such as *how to use this library?*, *what is the difference between two languages?* [1], or *how to understand this concept?* [2], [3], they tend to use Stack Overflow to find answers [4]. To assist developers in finding the knowledge they are looking for in such a large-scale knowledge repository, Stack Overflow provides a search engine[1], which supports free text search as well as an advanced search with metadata filters.

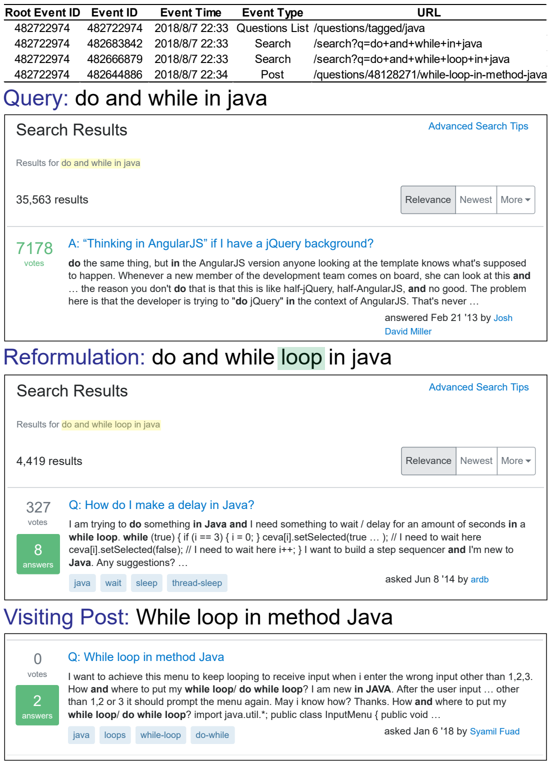
However, even using the provided search engine, it is still not easy for developers to effectively find what they want [5], [6]. There are two reasons for unsatisfactory search results. First, there exists a certain semantic gap between the users' query intention and input queries. This means that it is difficult for users to accurately express their query intention with a few keywords [7], [8]. For example, a developer wants to search for the usage of nested lists. However, they may not know how to express this concept accurately and may use a term like "*list in list*" as the query. This query is imprecise, and this kind of semantic gap can pose a significant challenge to the search engine of Stack Overflow. Second, a certain semantic gap exists between the users' queries and the text content in the relevant posts. It means the same meaning may be described in different ways with few overlapping words. For example, a developer may input the query "*sorting in linear time*". However, the relevant post titled "*sorting with $O(n)$ complexity*" cannot be retrieved by the search engine. Moreover, abbreviations, synonyms, or even misspelling [9], [10] can also lead to this kind of semantic gap.

To alleviate the above semantic gaps, developers may constantly reformulate their queries until the query reflects their real query intention and leads to relevant posts. By analyzing users' query logs provided by Stack Exchange Inc., the company behind Stack Overflow, we find that 24.62% of the queries are reformulated before visiting a post, and that developers reformulate their queries 1.46 times on average before clicking on a result. Fig. 1 shows an illustrative example. The user first performs the query "*do and while in java*", which yields a large number of irrelevant results. The reason is that the search engine uses *or* to connect all the words in the query, and the search engine cannot distinguish "*do*" and "*while*" as domain-specific keywords. Then, the user adds the word "*loop*" to this query to make it more explicit, which leads to a potentially desired post. The process of modifying a given query to find a satisfactory search result [11] is called query reformulation. Understanding query reformulation [12]–[16] has become an important issue in designing effective information retrieval systems.

To investigate how developers reformulate their queries, we first perform a formative study on the users' activity logs from Stack Overflow. Based on the logs from 2,216,219 users between December 2017 and November 2018, we extract 4,631,756 queries from 3,125,427 sessions. After analyzing these queries, we find that query reformulation has certain

* Corresponding Authors

[1] https://stackoverflow.com/search



Root Event ID	Event ID	Event Time	Event Type	URL
482722974	482722974	2018/8/7 22:33	Questions List	/questions/tagged/java
482722974	482683842	2018/8/7 22:33	Search	/search?q=do+and+while+in+java
482722974	482666879	2018/8/7 22:33	Search	/search?q=do+and+while+loop+in+java
482722974	482644886	2018/8/7 22:34	Post	/questions/48128271/while-loop-in-method-java

Query: do and while in java

Search Results

Results for do and while in java

35,563 results

7178 votes

A: “Thinking in AngularJS” if I have a jQuery background?

do the same thing, but **in** the AngularJS version anyone looking at the template knows what's supposed to happen. Whenever a new member of the development team comes on board, she can look at this **and** … the reason you don't **do** that is that this is like half-jQuery, half-AngularJS, **and** no good. The problem here is that the developer is trying to "**do** jQuery" **in** the context of AngularJS. That's never …

answered Feb 21 '13 by Josh David Miller

Reformulation: do and while loop in java

Search Results

Results for do and while loop in java

4,419 results

327 votes

8 answers

Q: How do I make a delay in Java?

I am trying to **do** something **in Java and** I need something to wait / delay for an amount of seconds **in** a **while loop**. **while** (true) { if (i == 3) { i = 0; } ceva[i].setSelected(true …); // I need to wait here ceva[i].setSelected(false); // I need to wait here i++; } I want to build a step sequencer **and** I'm new to **Java**. Any suggestions? …

asked Jun 8 '14 by ardb

Visiting Post: While loop in method Java

0 votes

2 answers

Q: While loop in method Java

I want to achieve this menu to keep looping to receive input when i enter the wrong input other than 1,2,3. How **and** where to put my **while loop**/ **do while loop**? I am new **in JAVA**. After the user input … other than 1,2 or 3 it should prompt the menu again. May i know how? Thanks. How **and** where to put my **while loop**/ **do while loop**? import java.util.*; public class InputMenu { public void …

asked Jan 6 '18 by Syamil Fuad

Fig. 1: An illustrative example of query reformulation in the users' activity logs

common patterns (in Section III-B). For example, users may fix misspellings, such as revising "*serive*" to "*service*"; they may generalize their queries to expand the scope, such as revising "*open calendar react native*" to "*open other app react native*"; they may add constraints of programming languages or platforms to the query, such as revising "*read file*" to "*java read file*"; they may remove information that is too detailed for retrieving relevant posts, such as revising "*Unable to import module 'copy': /var/task/psycopg2/_psycopg.so: ELF file's phentsize not the expected size*" to "*ELF file's phentsize not the expected size*".

As the query reformulation process is tedious for developers, especially for novices, we propose a **S**oftware-sp**E**cific **QUE**ry **R**eformulation approach (SEQUER) based on deep learning to help automatically reformulate their queries. Based on large-scale query logs from Stack Overflow provided under a non-disclosure agreement, we first extract query reformulation pairs consisting of original and reformulated queries and then adopt an attention-based Transformer to automatically learn the query reformulation patterns based on the extracted query reformulation pairs. Given the original query, the trained model can suggest a list of candidate reformulated queries. We evaluate the quality of the reformulated queries of our approach with large-scale archival manual reformulation results. The evaluation results show that, in terms of $ExactMatch@10$ and $GLEU$, our approach achieves 12.48% and 7.79% improvement on average compared with the sequence model based baselines (i.e., seq2seq with attention [17] and HRED-qs [18]), achieves 30.79% and 6.61% improvement compared with Google Prediction Service [19], and achieves 33.5% and 14.41% improvement compared with grammatical error correction tools.

In summary, we make the following contributions:

- We distill unique insight into developers' query reformulation patterns based on large-scale real-world query logs from Stack Overflow.
- According to the insights from our empirical study, we propose a software-specific query reformulation approach SEQUER based on an attention-based Transformer.
- We evaluate the quality of the reformulated queries generated by our approach SEQUER with large-scale archival manual reformulation results.
- We implement a browser plugin[2] for supporting automated software-specific query reformulation in practice.

II. Data Collection

The dataset we used is based on a larger dataset containing all internal HTTP requests processed by Stack Overflow's web servers within one year (747,421,780 requests from December 2017 to November 2018). Internal means that the dataset only contains requests with a referrer URL from `stackoverflow.com`. If a user, for example, reached a Stack Overflow post by clicking on a Google search result and then triggered a search within Stack Overflow, only the second (internal) search request would be included in the dataset, not the request for the post having a Google-related referrer. For each HTTP request, the dataset provides an anonymized user identifier that represents logged-in registered users as well as users identified by a browser cookie or users identified by their IP address. This dataset also assigns certain event types to the requests (e.g., searching, post visiting, or question list browsing), depending on their target URL.

Before extracting the event sequences relevant to this study, we preprocess the data as follows. First, we group all events per user identifier and then order them chronologically. Second, we utilize heuristics based on the timestamps and request targets to filter out bot traffic and event sequences merely consisting of page refreshes. Third, to distinguish between individual sessions, we group the events into sequences of events that are not more than six minutes apart, following Sadowski et al.'s approach [20]. Finally, we add an additional filtering step to avoid gaps in the data caused by the focus on internal requests. A user may, for example, follow external links in Stack Overflow posts and then navigate back to Stack Overflow or open multiple posts in parallel browser tabs. For our analysis of query reformulation on Stack Overflow, we focus on complete linear navigation sequence, that is sequence where the referrer of one request matches the target URL of the previous request.

After the above data preprocessing, we get a dataset of complete linear navigation sequences. As shown in the table in Fig. 1, each event is represented by one row with five attributes: *RootEventId*, *EventId*, *EventTime*, *EventType*,

[2] https://github.com/kbcao/sequer

and *URL*. Specifically, the *RootEventId* refers to the *EventId* of the first event in the session, the *EventId* is a unique ID that can identify an event, the *EventTime* indicates the UTC time when the event occurred, the *EventType* shows the type of the event (possible values of event type are {*Search*,*Post*,*QuestionsList*,*Home*,*Tags*,*PostHistory*}), the *URL* is the web request that triggers this event, which contains, for example, the query content or the post ID, depending on the *EventType*.

Our dataset contains 42,173,522 events from 16,164,506 sessions generated by 9,712,878 users, in which 9,046,179 events are queries. On Stack Overflow, the post visit event and the query event are the two most common operations performed by the users, accounting for 46.21% and 21.45% of the events respectively. Users often reach a post in three ways: links in the post (58.75%), search (17.25%), and question list (5.69%). That is, in addition to the navigation between posts, search is the most common way for the users to find a post.

From the session perspective, 30.82% of the sessions contain query event(s), and the users perform an average of 1.82 queries in these sessions. Fig. 2 shows the distribution of the number of sessions and their average session duration in terms of the number of queries contained in the session. In this figure, we can find as the number of queries in the session increases, the number of such sessions decreases exponentially, while the session duration increases linearly.

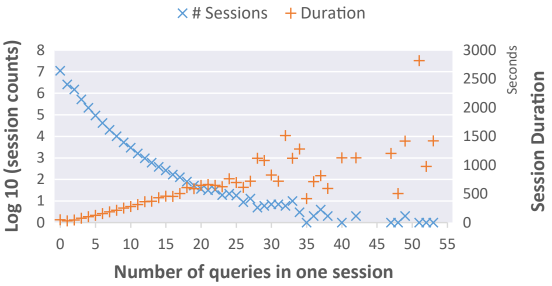


Fig. 2: The distribution of sessions number and duration in terms of the number of queries contained in the session

To train our query reformulation model, we obtain the query reformulation records (i.e., the user's process of transforming an original query into a better one) from the user's navigation sequence. We call these processes the query reformulation threads. We first remove the sessions that do not contain a query event or of which the last event is not a post visit. The latter limitation is to ensure that the users finish the query reformulation with a desired results. This yields a dataset containing 8,546,915 events from 3,125,427 sessions generated by 2,216,219 users, in which 4,631,756 events are queries. We adopt a pattern-based method to extract the query reformulation threads from each session following a greedy approach:

$$\cdots, q_1, q_2, \cdots, q_i, p_1(optional), q_{i+1}, \cdots, q_n, p_m \cdots$$

In this pattern, we use q_i to denote the i-th query event and use p_i to denote the i-th post visit event. All the events are ordered chronologically. We conjecture that q_n is a relatively better query compared to the queries from q_1 to q_{n-1}. Sometimes users may need to visit posts in the query results to determine whether a particular result is what they want, and they may reformulate their queries again after visiting those posts to get better results. Therefore, to make the pattern more versatile and avoid misrecognition of reformulated queries, post visit event is allowed in the sequence of query events. More details on extracting query reformulation threads can be found in Section IV-B.

III. Empirical Study of Query Reformulation on Stack Overflow

In this section, to motivate the required tool support, we perform a formative study to understand the characteristics of query reformulation by analyzing the Stack Overflow log data.

A. What are the characteristics of queries?

Query content analysis. We analyze the query strings to investigate what the users are searching for on Stack Overflow. First, we collect all query strings and apply traditional text processing steps (i.e., removing punctuation, transforming to lower case, excluding stop words, stemming) to them. Then, we identify the most popular n-grams in the queries.

Table I shows the top-10 most frequent 1-grams, 2-grams, 3-grams, and 4-gram in the users' queries. Programming languages such as *Python* and *Java*, platforms such as *Android*, data types such as *string*, and data structures such as *array* are the most frequently queried terms. At the same time, *"how to"* is the most frequently used phrase in the queries. Almost every 3-gram starts with *"how to"*, next comes *"what is"* and programming language qualifiers such as *"in python"* and *"in java"*. It is worth noting that some *Java* and *Python* error logs appear in the top ten 4-grams. The reason is that the developers often paste these error logs directly into the search box to perform queries, and logs like "exception in thread "main"" and "ImportError: No module named" are the most common error types.

TABLE I: The top-10 most frequent n-grams in the users' queries

Rank	1-gram	2-gram	3-gram	4-gram
1	python	how to	how to use	how to create a
2	java	in python	how to get	how to make a
3	file	what is	how do i	how do i use
4	string	in java	how to create	exception in thread "main"
5	android	failed to	how to make	ImportError: No module named
6	c#	in swift	could not find	how to get the
7	error	in r	how to change	no such file or
8	array	unable to	how can i	how to add a
9	sql	how do	how to find	how to check if
10	list	not found	how to install	such file or directory

Query length analysis. The users may intentionally limit a query's length to avoid returning a few or even empty results when using traditional search engines [21]. To investigate whether this phenomenon exists when the users perform queries on Stack Overflow, we use whitespaces as the separator to compute the query length (i.e., word count). Note that we treat words synthesized by CamelCase or underscore_case as one word since these words often appear in code snippets and can be regarded as the identifiers of variables, classes, or methods.

Fig. 3 shows the distribution of query length via a box plot. The median, mean, 25^{th} percentile, and 75^{th} percentile of the length are 3, 3.6, 2, and 4 respectively. The distribution shows that users do intentionally limit the length of their queries to get better search results. However, we can also easily find that the query length span of the outliers is very large, and all values from 8 to 25 correspond to outliers. For these queries, it is difficult for the search engine of Stack Overflow to return satisfactory results. Because most search engines are optimized only to handle common requests, they use exact-match techniques in which all query words must match a web page for web page retrieval [22]. A longer query means lower matching probability and leads to lower-quality search results. After manual analysis, we find that most of these queries contain error messages or code snippets.

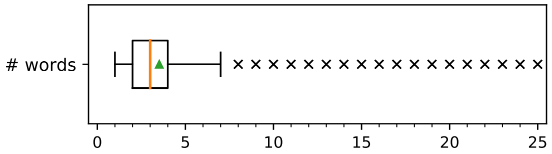


Fig. 3: The distribution of query length by using box plot (note a few outliers with more than 25 words are removed)

Advanced search methods analysis. We notice that some of the queries in the log are structured, which means the users use advanced search methods[3]. Advanced search methods can provide users with a convenient way to filter search results. For example, the users can narrow the search results by only considering the posts with the tag *python* or the posts that have a minimum score of 500. Stack Overflow provides 29 advanced search methods to help the users filter search results. We first apply regular expressions to the users' queries to identify the used advanced search method(s). Then we calculate the ratio of the queries using advanced search and the proportion of each advanced search method.

The result shows that 8.74% of the queries use the advanced search. The top-10 most frequently used advanced search methods, an illustrative example, and their proportions can be found in Table II. Tag filtering, user filtering, and declaring specific phrases are the top-3 most commonly-used advanced search methods, accounting for 73.53%, 10.35%, and 10.00% respectively.

B. Why are queries reformulated?

The users reformulate their queries for many reasons. For example, the queries may not specify the corresponding programming language, or some words in the queries may be misspelled. We use the provided dataset to analyze why the users reformulate their queries. We manually classify 384 randomly collected query reformulation threads[4] into four categories. The first and second author classified these 384 threads independently. For the cases without an agreement, the final classification result is determined through discussions. For the threads with more than two queries, if multiple query reformulation types are identified, the type of reformulation from the first query to the last query is used as the final classification result. The Kappa inter-rater agreement [24] is 0.83, which shows the high agreement of classification.

TABLE II: The top-10 most frequently used advanced search methods

Type	Example	Proportion
tag	**[powershell]** job output	73.53%
user	**user:**8945947	10.35%
declare specific phrase	ios **"save to files"**	10.00%
exclude phrase	accordion **-jquery**	3.88%
Wildcard	**[xamarin*]** does not support...	0.74%
question only	**is:question** powershell version	0.44%
#answers	**answers:0** firebase	0.28%
multiple tags	**[scipy] or [numpy]** array vs matrix	0.26%
score	safari cache **score:3**	0.19%
creation date	oauth read gmail **created:05-04-2015..**	0.06%

Table III shows the manual classification results. In this table, we can see that adding new information to the query is the most common query reformulation operation, which accounts for 40.1% of all threads, then comes modifying the query (33.59%), and deleting information from the query (23.18%). In the category of adding new information [25], we further divide it into two sub-categories: (1) adding information about specific programming languages or platforms, (2) adding new requirements for the question or more detailed content. In the category of modifying, we further divide it into three sub-categories: (1) spelling and syntax checking [26], (2) simplifying and refining the query to reformulate it into the most commonly-used expression, (3) modifying to other content related to the original query. In the category of deleting, we further divide it into three sub-categories: (1) deleting some unnecessary or less informative words, (2) deleting specific information in error messages or code snippets (such as file path, URL, function names, etc.), (3) deleting punctuation and some mistyped symbols.

Based on the classification result, we can observe many different types of query reformulation operations (such as adding information, deleting information, and modifying their expressions). Previous studies [27]–[30] mainly use rule-based methods to perform automated query reformulation. However, each of these methods can only target query reformulation for a specific reason. Different rules need to be designed for each situation and then implemented with different methods, which is inefficient and difficult to achieve. For example, for the query reformulation in the category of modifying,

[3] https://stackoverflow.com/help/searching

[4] The number is the minimum number to be statistically representative of a large dataset with a confidence level of 95% and error margin of 5% via a commonly-used sampling method [23].

TABLE III: Categories of query reformulation

Category	Sub-category	Example	Proportion
Add	Software or platform	why to use sha1 => why to use sha1 **in android**	21.61%
	Detailed requirement	db file => **open** db file	18.49%
		Category subtotal	40.10%
Modify	Spelling and syntax check	.net string.**emptyp** => .net string.**empty**	19.01%
	Simplify and refine	**list inside list** for sightly => **nested list** in sightly	12.50%
	Turn to related information	python program **freezes** => python program **hangs**	2.08%
		Category subtotal	33.59%
Delete	Detailed or unnecessary words	C# update a **keypairvalue** in a Dictionary => C# update Dictionary	15.10%
	Specific information in error message	**Property ’getData’** does not exist on type ’ReactInstance’ => does not exist on type ’ReactInstance’	6.51%
	Symbols or web links	**:=** dbms_datapump.open => dbms_datapump.open	1.56%
		Category subtotal	23.18%
Others			3.13%

the user may completely change the expression of a query (e.g., from “*how to cut youtube embedded videos*” to “*how to make embedded videos that only play certain parts*”). It is challenging to implement this type of reformulation by using rule-based methods. Therefore, we find it necessary to propose a general query reformulation approach, which is the motivation of this study.

C. What is the scale of changes that query reformulation involves?

In Section III-B, we find that when reformulating a query, the users may add, remove, or replace some words in the original query. To understand the scale of changes that query reformulation involves, we measure the similarity between the original query and the reformulated query. For each query reformulation thread, $n-1$ pairs of reformulation samples (*original*, *reformulated*) can be extracted, that is $\{(q_1, q_n), (q_2, q_n), \cdots, (q_{n-1}, q_n)\}$. The first query *original* in the pair is the initial query performed by the user, and the second query *reformulated* is the user’s manual reformulated query, which meets their requirement. We use a text-matching algorithm with improved dynamic programming [31] to find the character-level Longest Common Subsequence (LCS) between *original* and *reformulated*. Then the similarity between *original* and *reformulated* can be defined as follows:

$$similarity(original, reformulated) = \frac{2 \times N_{match}}{N_{total}} \quad (1)$$

where N_{match} is the number of characters in the LCS, and N_{total} is the sum of the number of characters in *original* and *reformulated*. The similarity score is in the range of 0 to 1. The higher the similarity score, the fewer changes between *original* and *reformulated*.

As shown in Fig. 4, among 1,121,185 query reformulation pairs, 58.07% of the pairs are very similar (i.e., the similarity score is larger than 0.7) with an average of 6.73 character level modifications, which corresponds to 1.14 words according to the average length of query words. The analysis result indicates that most of the query reformulations only involve minor changes.

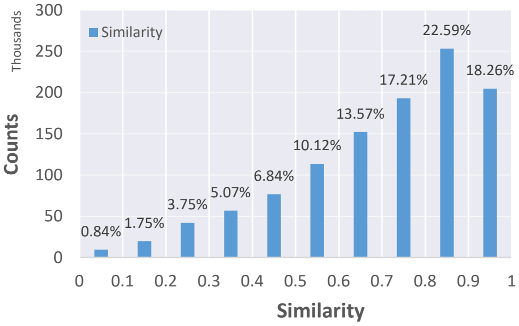


Fig. 4: Distribution of the similarity score of query reformulation pairs

D. Summary and implications

Our empirical study shows that: (1) On Stack Overflow, most of the users’ queries are simple and short with only 2 to 4 words, but there are also very long queries containing error logs, code snippets, and structured queries with advanced search patterns. (2) The users perform query reformulation for various reasons, making it difficult to design a general query reformulation model by only using rule-based methods. (3) Most query reformulations only involve minor changes.

Considering the diversity of query reformulation patterns, it would require significant effort to manually develop a complete set of reformulation patterns, which is both time-consuming and error-prone. Unlike enumerating all the rules, the deep learning method [32] can automatically learn the latent features from the dataset and these latent features are helpful for downstream tasks.

Based on the above findings, we believe it is necessary and feasible to propose an automated query reformulation approach based on deep learning. Our proposed approach would benefit the developers and the Stack Overflow community. In particular, the developers can use our automated query reformulation tool to refine their queries and get better search results, which can alleviate the effort of manual reformulation. The Stack Overflow community can also use the query reformulation model to enhance the users’ search experience.

IV. Automated Query Reformulation

A. Overview of SEQUER

Our formative study has shown a wide variety of query reformulation behaviors. Since not all reformulation patterns are amenable to automated tool support, we focus on query reformulations that trigger minor changes to the original query to develop our automated query reformulation approach. In this work, we formulate query reformulation as a machine translation problem, in which an original query is "translated" into a reformulated query. We solve the problem using the Transformer model [33].

The overall workflow of SEQUER is shown in Fig. 5. SEQUER first extracts query reformulation threads for sampling original queries and corresponding reformulated ones. Then a large corpus of query reformulation pairs is gathered for training the model, each of which contains the original query and the corresponding reformulated one that triggers only minor changes (in Section IV-B). To model the patterns of query reformulation (such as spelling correction, expression refinement, unnecessary word deletion), SEQUER trains a Transformer-based model with a large parallel corpus of query reformulation pairs (from Section IV-C to Section IV-E). Given the users' original queries, the trained model can suggest a list of reformulated candidates for selection.

B. Collection of query reformulation pairs

Based on the description in Section II, query reformulation threads can be extracted from the users' navigation sequences by pattern matching. The most important thing for the pattern is how to ensure that q_i and q_{i+1} are issued for the same purpose, rather than two independent queries. Two constraints are applied: (1) The character-level similarity between q_i and q_{i+1} must be greater than 0.7 by using Equation (1). (2) The browsing time of the post p_1 cannot be greater than 30 seconds. The first constraint can limit the change scale since a large-scale change is likely to introduce a new query. The second constraint can guarantee the post p_1 is not what the user wants, since a previous study [34] shows for an unsatisfactory post, the user would not spend more than 30 seconds on it.

We set the minimum number of query events before the final post visiting event in the thread to 2. After identifying all the query reformulation threads, we perform the following data cleaning steps. For consecutive identical queries in the thread, we only keep one of these identical queries and remove the rest to avoid generating duplicate query reformulation pairs. We also remove threads containing queries with non-English words, since non-English words are not encouraged to be used on Stack Overflow[5]. After the above data cleaning, we pair each query between q_1 to q_{n-1} with q_n (i.e., (q_1, q_n), (q_2, q_n), $\cdots$, (q_{n-1}, q_n)) as the training instances. From the provided dataset, we extract a total of 1,121,185 query reformulation pairs.

Some of these reformulation pairs are hard-to-predict due to their large-scale modifications. After randomly sampling reformulation pairs with different similarity, we carry out a pilot study to determine the similarity threshold, which can be used to identify large-scale modifications. The results show that reformulation pairs with a similarity score higher than 0.7 are predictable, while the remaining pairs (i.e., reformulating by large-scale modifications) cannot be predicted even by humans. As analyzed in Section III-C, 58.06% of the query reformulations involve minor modifications with a similarity of at least 0.7. Therefore, we only consider these similar-enough query reformulations, which results in 651,036 query reformulation pairs.

C. Byte pair encoding

Traditional word segmentation methods include whitespace-separated word level and character level segmentation. However, word-level embedding cannot handle the Out-Of-Vocabulary (OOV) problem [35], and the character level is too fine-grained, resulting in the loss of high-level information. Therefore, we adopt the BPE (Byte Pair Encoding) [36], which can effectively interpolate between word-level inputs for frequent symbol sequences and character-level inputs for infrequent symbol sequences.

The BPE algorithm contains multiple iterations. In each iteration, it calculates each consecutive byte pair's frequency and finds the most frequent one, then merges the two byte pair tokens into one token. This encoding approach can divide words into sub-words like encoding the common words at the word level while encoding the rare words at the character level. Since the form of the users' queries on Stack Overflow varies, using BPE can better identify the content of the users' queries. For example, misspelled words can be divided into several correctly spelled sub-words to alleviate the OOV problem's impact. Words with camel-case style can be separated into several sub-words and then each sub-word can be identified. Besides, using BPE to separate words with their affixes can help the model learn the relationships between them. After applying the BPE algorithm on the corpus, we get a vocabulary composed of sub-words, which is used as the dictionary for our model.

D. Transformer

With the introduction of the attention mechanism [37], many neural machine translation approaches integrate the attention mechanism with sequence transduction models like Convolution Neural Network (CNN) and Recurrent Neural Network (RNN) to improve their performance. Even so, the CNN-based network architectures require many layers to capture long-term dependencies, leading to high computational cost, and operations in RNN-based network structures cannot be parallelized, resulting in low efficiency. To address these issues, Transformer [33] is proposed, which is the first transduction model entirely relying on the attention mechanism and has shown competitive performance on various tasks (such as machine translation [33], image captioning [38], document generation [39], and syntactic parsing [40]).

[5] https://stackoverflow.blog/2009/07/23/non-english-question-policy/

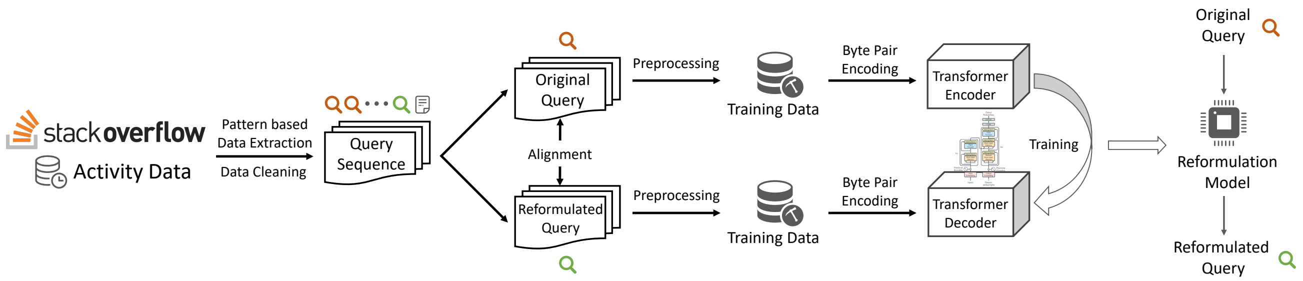


Fig. 5: The overall workflow of our approach SEQUER

The most significant difference between Transformer and other sequence transduction models like CNNs and RNNs is that it relies entirely on self-attention (called "Scaled Dot-Product Attention") to obtain global dependencies, and the attention weights are defined by how each word of the sequence is influenced by all the other words in the sequence. The self-attention mechanism creates shortcuts between the current token and all the other context tokens to determine the current token vector for the final input representation. As weights of these shortcuts are customizable, the self-attention mechanism is able to capture global dependencies without using many layers of convolution and pooling in CNN-based models. At the same time, the calculation in self-attention is implemented with highly optimized matrix multiplication, and thus resolves the low efficiency caused by the RNN-based models, which sequentially encode the input tokens. Intuitively, for an input sequence, first, a neural network is employed to map the input into three matrices: query Q, key K, and value V. Then, the dot products of the queries Q with all keys K is divided by $\sqrt{d_k}$ (d_k is the dimension of the queries), and a softmax function is applied to obtain the weights for the values. Finally, the weighted value is used as the representation of each input.

$$\text{Attention }(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{2}$$

Instead of performing a single self-attention function, the Transformer employs a multi-head self-attention, which linearly projects the queries Q, keys K and values V h times with different, learned linear projections respectively, where h is the number of heads. To obtain the temporal relationship of the words, the Transformer adds positional embedding to the input embedding.

Self-attention can be regarded as a basic calculation in Transformer. The Transformer model comprises an encoder and a decoder, which are actually multiple identical encoder and decoder blocks stacked on top of each other with the same number of units. Each encoder block has one layer of multi-head self-attention followed by another layer of Feed Forward Network (FFN). On the other hand, each decoder has an extra masked multi-head self-attention, which prevents the model from seeing the generated words during parallel training. On the encoder side, the multi-head self-attention layer's output is the input embedding with temporal information, and the layer output is normalized and sent into an FFN, which consists of two linear transformations with a ReLU activation. The output of the encoder on the top of the stack is a set of attention vectors K and V, which are used by the decoder to determine the token it should pay attention to. On the decoder side, the previous output is used as the input to the masked multi-head self-attention layer. After that, another multi-head self-attention layer with subsequent FFN generates decoder output h_t by getting the query matrix Q from the masked multi-head self-attention layer, the key K and value V matrices from the output of the encoder stack. Finally, the output of the decoder stack h_t is sent to a fully connected neural network to get the logits vectors, and then a softmax layer to predict the probabilities of the next token.

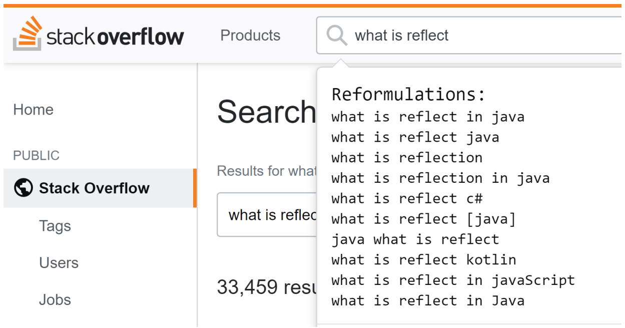
$$P(w_{t+1}|w_1, \cdots, w_t) = \text{softmax}(h_t W + b) \tag{3}$$

where h_t is the output of the decoder stack.

E. Beam search

During decoding, for each time step t, the Transformer model will output the word with the highest conditional probability $y_t = \text{argmax}_{y \in \mathcal{D}} P(y|y_1, \cdots, y_{t-1})$ via greedy search from $|\mathcal{D}|$ number of words, where $\mathcal{D}$ represents all the words in the word dictionary. Since we calculate the conditional probability of generating an output sequence based on the input sequence $\prod_{t=1}^{T} P(y_t|y_1, \cdots, y_{t-1})$, where T is the maximum length of the output sequence, the main problem with greedy search is that there is no guarantee that the optimal sequence will be obtained. The reason is that although the greedy strategy ensures that the output candidate with the highest probability is picked up for each time step, it cannot ensure that the conditional probability of the entire output sequence obtained is the highest.

Beam search [41] is an improved algorithm of greedy search, and beam size k is its hyper-parameter. The decoding process using beam search is as follows: At time step 1, k words with the highest probability are selected as the first word of k candidate output sequences cs_1. Then, at time step i, k output sequences with the highest conditional probability cs_i will be selected from $k|\mathcal{D}|$ possible output sequences based on cs_{i-1}. After T iterations, k sequences with the highest score will be selected from CS as the beam search result, where T is the maximum number of tokens of the output sequence, and CS is the collection from cs_1 to cs_i. Note, for each sequence, portions including and after special end-of-sequence tokens are discarded. The score is calculated as follows:



```
Reformulations:
what is reflect in java
what is reflect java
what is reflection
what is reflection in java
what is reflect c#
what is reflect [java]
java what is reflect
what is reflect kotlin
what is reflect in javaScript
what is reflect in Java
```

Fig. 6: A screenshot of our query reformulation plugin for the search engine of Stack Overflow

$$\frac{1}{L^{\alpha}} \log P\left(y_1, \cdots, y_L\right) = \frac{1}{L^{\alpha}} \sum_{t=1}^{L} \log P\left(y_t | y_1, \cdots, y_{t-1}\right) \tag{4}$$

where L is the length of the sequence in CS and α is the length normalization parameter.

F. Implementation

In our implementation, the maximum vocabulary size for BPE [36] is set to 10,000. For the Transformer, we use the tensor2tensor library [42] developed by the Google Brain team. The Transformer model contains four attention heads with four encoder and decoder layers, with *hidden_size* $= 512$. During the model training, the parameters are learned by back propagation [43] with Adam optimizer [44] to minimize the error rate. We train our model with *batch_size* $= 256$, *learning_rate* $= 0.0001$ for 147 epochs on 4 Nvidia V100 GPU (32G memory) for about 8 hours. During decoding, the hyper-parameter k of beam search is set to 10 to ensure the probability of finding the optimal solution. The length normalization parameter α is set to 0.6, which is a common practice in neural machine translation [45].

To make our work more practical, we develop a browser plugin[6] based on Tampermonkey, which is a popular user-script manager. The browser plugin will automatically analyze the query content and recommend the top-10 query reformulation candidates to the users for selection (a screenshot can be found in Fig. 6). Although the plugin is designed to only work on Stack Overflow now, it can be easily extended to other software-specific Q&A sites.

V. Quality of Recommended Query Reformulation

A. Dataset

From the 651,036 query reformulation pairs, we randomly take 520,830 (80%) of these query pairs as the training set to train the model, 65,103 (10%) as the validation set to tune model hyper-parameters, and 65,103 (10%) as the testing set to evaluate the quality of recommended reformulations.

B. Baselines

As analyzed in Section III-B, many queries are reformulated to fix grammatical errors. Therefore, we first adopt a popular grammatical error correction (GEC) tool as the baseline. LanguageTool[7] is an open-source proof-reading tool for more than 20 languages. This tool's style and grammar checker is rule-based and has been developed for over ten years.

As our task is query reformulation, and Google is the most popular search engine in the world, we choose Google Prediction Service (GooglePS) [19] as a baseline. Google uses a prediction service to help complete searches in the search box or address bar within Chrome. These suggestions are based on the real searches that happen on Google. Common and trending queries relevant to the strings entered by the users are shown in the drop-down bar for the users to choose[8]. Since Google has accumulated a large-scale dataset of the users' queries, GooglePS can efficiently and accurately reformulate users' queries, such as correcting misspelled words, completing words that the users are typing, and appending the next possible word.

The query reformulation task can also be regarded as a translation task (i.e., translating the original query into the reformulated query). Therefore, we take the most classical neural machine translation model seq2seq [17] as our baseline. It contains one Long Short-Term Memory (LSTM) model [46] as the encoder, which can encode the original query to an embedding vector, and another LSTM model, which can decode that embedding vector to the reformulated query. We also add the seq2seq model with the attention mechanism as another baseline.

Besides, there are many studies in information retrieval [47]–[50] about query suggestion. We select HRED-qs [18] as one of our baselines, which is a representative hierarchical and session-based query suggestion model with full source code release. HRED-qs is trained with our dataset. In particular, given a query in the session, HRED-qs first encodes the information seen up to the position by a query-level RNN encoder and a session-level RNN encoder. Then it uses the following decoder to predict the next query. To keep the setting of all baselines consistent, we only feed the last query before the reformulated one in the session to HRED-qs.

To make a fair comparison, we employ the same data preprocessing method and Byte Pair Encoding for seq2seq and HRED-qs as SEQUER, and we perform hyper-parameter optimization with grid search [51].

C. Evaluation metrics

Query reformulation is similar to the grammar error correction task (i.e., revising some words in the original sentence for generating the target sentence). Therefore, we evaluate the query reformulation with the metrics used in the GEC task (i.e., $GLEU$, M^2 and $ExactMatch$).

[6] https://github.com/kbcao/sequer

[7] https://languagetool.org

[8] https://www.blog.google/products/search/how-google-autocomplete-works-search

$GLEU$ (General Language Evaluation Understanding) [52], [53] is a customized metric from $BLEU$ (BiLingual Evaluation Understudy) [54], which is a widely used metric to measure the performance of machine translation approaches. Since only part of the source sentence will be changed in the GEC task, which is different from the machine translation task, this motivates a small change to $BLEU$ that computes n-gram precision over the reference but assigns more weight to n-grams that have been correctly changed from the source. Therefore, compared with $BLEU$, $GLEU$ is more suitable for evaluating query reformulation in our study.

$MaxMatch$ (or M^2) [55] is another widely used GEC evaluation metric that computes the sequence of phrase-level edits between a source sentence and a system hypothesis that achieves the highest overlap with the gold-standard annotation. These edits are scored by $precision$, $recall$, and *F1*. Specifically, in the scenario of query reformulation, $M^2@P$ represents the proportion of edits of the original query given by an approach that appears in the user's manual reformulation. $M^2@R$ represents the proportion of edits of the original query by users that are correctly predicted by an approach. $M^2@F1$ is the harmonic mean of $M^2@P$ and $M^2@R$.

$ExactMatch$ (EM) evaluates the probability of a perfect match between the query provided by a specific approach and the user's manually reformulated one. Since SEQUER uses beam search during decoding, it can suggest multiple reformulations for an original query, for the decoding results of SEQUER with different beam size, we can calculate *EM@1*, *EM@5*, and *EM@10*, where *EM@n* means one case will be considered positive as long as one of n reformulation results returned by beam search matches the ground truth.

D. Evaluation results

We report our evaluation results by answering the following two research questions.

1) RQ1: Can our approach SEQUER generate better reformulated queries than the baselines?: Table IV shows the evaluation results of query reformulation on the testing set of SEQUER and baselines in terms of all the evaluation metrics. SEQUER outperforms all state-of-the-art baselines by significant margins in terms of all metrics. Compared with the best baseline seq2seq (with attention), SEQUER achieves 5.6% and 4.75% improvement in terms of *EM@10* and $GLEU$. To better illustrate our results, Table V lists examples of query reformulations by different approaches, and more examples can be found on our project site[9].

TABLE IV: Performance of automated query reformulation

Approach	EM@1	EM@5	EM@10	GLEU	M^2@P	M^2@R	M^2@F1
LanguageTool	—	—	5.87	53.27	15.31	6.11	8.73
GooglePS	6.20	8.14	8.58	61.07	24.67	25.84	25.24
HRED-qs	10.40	17.63	20.01	56.85	31.80	25.82	28.50
seq2seq	11.02	23.11	28.23	61.30	36.30	25.62	30.04
seq2seq+Attn.	14.53	28.47	33.77	62.93	35.93	21.15	26.62
SEQUER	**22.21**	**33.47**	**39.37**	**67.68**	**39.67**	**31.97**	**35.41**

GEC tools perform well in correcting misspellings ("*strin*" to "*string*" in Example 1), grammar issues ("*playing*" to "*play*" in Example 2), and sentence format ("*currentdate*" to "*current date*" in Example 3). However, they only achieve a 5.87% exact match, as spelling errors only account for 19% of the reasons for the users' query reformulations (in Section III-B). In addition, they have difficulty in detecting spelling errors specific to the programming field. For example, they cannot detect any misspelled words in "how to import bumpy array" where the word "*bumpy*" is the wrong spelling of "*numpy*" (a *Python* library). Instead, SEQUER can correct these software-specific misspelled words by learning the domain knowledge from our software-specific dataset (another example is from "*C3*" to "*C#*" in Example 4).

GooglePS can perform complex reformulations by learning from billions of queries Google processes every day, such as adding important missing keywords ("*loop*" in Example 5) and suggesting language/platform limitations ("*python*" in Examples 1 and 6). However, as a general-purpose search engine, Google does not perform well in software-specific query reformulation, especially for unpopular software-specific queries. For example, GooglePS cannot reformulate the query "/usr/bin/ld: skipping incompatible libpthread.so" in Example 7, but SEQUER can remove the file directory to make it more general. As the file directory often varies between developers' coding environments, it should be removed to keep the query more general, for retrieving more accurate results. By learning from massive query reformulations, which are software-specific from Stack Overflow, SEQUER can effectively revise such queries by applying software-specific reformulation strategies (similar examples in Examples 8 and 9).

Since using the same training data as SEQUER, the HRED-qs, seq2seq and seq2seq with attention model can capture software-specific semantics during query reformulation. This is the reason why they achieve better performance than the other baselines. However, since the goal of HRED-qs is basically "next query prediction", the context-aware hierarchical encoding does not enhance the ability of the decoder to generate a reformulated query, but may obscure the information of the original query, causing a deviation of the decoder result. For example, "comboox lost focus" should be reformulated to "combobox lost focus", but HRED-qs reformulates it to "iphone x lost focus". Besides, the problem for both HRED-qs and seq2seq is that the input is encoded into one single vector representation, which may not be sufficient to store all the information, especially for long queries. For example, "Allow user to paste url with .ph and rpeturn clean url" should be reformulated to "Allow user to paste url with .php and return clean url". However, seq2seq model can only reformulate it to "Allow user to paste url with php" due to the query length. On the contrary, SEQUER does not have this problem by using an attention-based Transformer model.

In addition to the types of query reformulations mentioned above, SEQUER also outperforms baselines in more complex reformulations such as revising with more commonly-used

[9] https://github.com/kbcao/sequer

TABLE V: Examples of query reformulation by different approaches ("—" represents no reformulation suggestions)

ID	Original Query	GooglePS	seq2seq + Attn.	SEQUER
1	pandas delete last characters in strin	**python** delete last characters in **string**	pandas delete last characters in **string**	**python** delete last characters in **string**
2	playing sound in swift3	playing sound in **swift 3**	—	**play** sound in **swift 3**
3	swift grab currentdate	—	swift ~~grab~~ **current date**	swift **get current date**
4	assign string to number C3	—	assign string to number **C#**	assign string to number **C#**
5	do and while in java	do and while **loop** in java	—	do and while **loop** in java
6	requests negotiate	requests negotiate **python**	python requests negotiate	**[python]** requests negotiate
7	/usr/bin/ld: skipping incompatible libpthread.so	—	**lib**skipping incompatible libpthread.so	~~/usr/bin/ld:~~ skipping incompatible libpthread.so
8	subtracting the pandas series rf from all columns	—	subtracting the pandas series ~~rf from all columns~~	subtracting the pandas series ~~rf~~ from all columns
9	<trigger>Missing report definition</trigger>	—	~~<trigger>~~Missing report definition</trigger>	~~<trigger>~~ Missing report definition ~~<trigger>~~
10	opencv scale image	opencv scale image **to 0 1**	—	opencv **resize** image
11	a* search	a* search **example**	a* search **python**	a~~*~~ **star** search
12	df -h show disk	—	—	**"df -h"** show disk
13	resteasy how to support websocket	—	resteasy ~~how to support~~ websocket	resteasy ~~how to support~~ websocket
14	volume control programatically android	**android** volume control **programmatically**	—	volume control android

software-specific terms (e.g., "swift grab currentdate" to "swift get current date" in Example 3, "opencv scale image" to "opencv resize image" in Example 10), replace symbols with text (e.g., "a* search" to "a star search" in Example 11) or enclose symbols that will accidentally trigger advanced search in quotation marks (e.g., "df -h show disk" to ""df -h" show disk" in Example 12 to prevent showing results for "*df show disk*" and not containing "*h*"), and simplify or refine the query (e.g., "resteasy how to support websocket" to "resteasy websocket" in Example 13 and "volume control programatically android" to "volume control android" in Example 14).

2) RQ2: What types of query reformulations are challenging for SEQUER to deal with?:

Although SEQUER can achieve the best performance compared to state-of-the-art baselines, SEQUER also makes mistakes in some query reformulations. We manually check some randomly sampled erroneous reformulations and identify two main reasons why our reformulations do not match the ground truth.

First, some queries are edited to add more information, which is beyond the context of the original query such as "python covert variable to integer" to "python convert hexadecimal to integer" and "git commit -am" to "git commit -am vs git add". Although SEQUER can successfully revise the misspelling "*covert*" to "*convert*", it cannot guess the replacement of "*variable*" with "*hexadecimal*" or adding "*vs git add*" by considering only the local context of the query. To support such complicated reformulation, we need to consider the broader context of the search (e.g., previous queries) in the future.

Second, the same meaning may be expressed in different ways. For example, given the original query "remove , from input", SEQUER recommends the reformulated query as "remove comma from input", however, the ground truth is "remove "," from input". Similarly, given the original query "read mouse cursor", SEQUER recommends the reformulated query as "c# read mouse cursor", however, the ground truth is "read mouse cursor in C#". Although the users' reformulation results and our recommendations are not exactly matched, they convey the same meaning. Some of our recommendations are of higher quality than the users' reformulation and may lead to better search results. That is also why the performance of SEQUER is highly underestimated.

E. Discussion

In Section V-D, we evaluate the quality of reformulations from different approaches by comparing them with users' manual ones with the metrics used in the GEC task. However, none of these metrics consider the model effectiveness (i.e., the ability of the reformulated query to retrieve the desired post). Therefore, we further evaluate the effectiveness of SEQUER in this section.

For each query reformulation thread (mentioned in Section II), we collect pairs of users' original query and finally-visited post, which is assumed to be the target post. These query-post pairs can be used to demonstrate the retrieval effectiveness of our approach. For each pair, we adopt both our approach and baselines in Section V-B to reformulate the query and check the ranking of the target post in all candidate posts on Stack Overflow. All the 65,103 queries in the testing set are paired with their corresponding finally-visited post as the evaluation data.

We use *MRR* (Mean Reciprocal Rank) as the metric for effectiveness evaluation. *MRR* is the average of the reciprocal ranks (i.e., the multiplicative inverse of the target post's rank in the search result) of the search results for all the queries. For example, given a query, if the target post ranks fifth in the search result, the reciprocal rank is 1/5=0.2. A higher *MRR* value indicates a better search result.

The comparison results between SEQUER and baselines in terms of *MRR* can be found in Table VI. SEQUER achieves the best performance (i.e., 129.33% boost in terms of *MRR* to the original query). Even compared with the best baseline seq2seq+Attn., SEQUER still achieves a 23.7% boost. This result shows that the reformulated query given by SEQUER is not only the closest to manual reformulation, but also has the best retrieval effectiveness among all the baselines. Therefore, SEQUER can effectively help users obtain better search results via high-quality query reformulation.

TABLE VI: Evaluation result of retrieval effectiveness

Approach	*MRR*	Boost Rate
Original Query	0.075	—
LanguageTool	0.071	5.33% ↓
GooglePS	0.083	10.67% ↑
HRED-qs	0.108	44.00% ↑
seq2seq	0.127	69.33% ↑
seq2seq+Attn.	0.139	85.33% ↑
SEQUER	**0.172**	**129.33%** ↑

VI. Related Work

Information Retrieval (IR) has been widely used for software engineering (SE) tasks, such as traceability recovery [56],

[57], feature location [58], [59], library migration [2], [60], [61], API search [62], [63] and GUI design seeking [64]–[66]. In this section, we summarize the related works about query reformulation in general IR and its application in SE domain.

A. Query reformulation in general information retrieval

To help users better refine their queries, there are many studies on query expansion [67], [68], query reformulation [69], [70] and query suggestion [47]–[50] in the information retrieval domain. In detail, Jiang et al. [48] tried to provide query suggestions based on previous queries in the session by introducing a binary classifier and an RNN-based decoder as the query discriminator and the query generator. Chen et al. [49] proposed an attention-based hierarchical neural query suggestion model that combines a session-level neural network and a user-level neural network to model the users' short and long term search history.

However, most of these studies are session-based or user profile-based and apply to general text search. Different from general text search, search in the software engineering domain is very specific, with domain-specific terms and code snippets, which make general approaches not applicable in this context. Therefore, we carry out this study based on domain-specific dataset for providing a software-specific automated query reformulator.

B. Query reformulation for document search in SE

The performance of document search in software engineering relies on the domain-specific query reformulation. Haiduc et al. tried to evaluate whether query reformulation is needed. They propose several metrics to measure or predict query difficulty [71], query specificity [72], and query quality [73] for concept location. Based on these metrics, they [74], [75] further developed a machine learning model to adopt one of four strategies to recommend corresponding revised queries. Rahman et al. [76] proposed a word-embedding based method to extract semantically similar terms, hence suggesting semantically relevant queries with terms from programming questions on Stack Overflow. Li et al. [77] shared a similar idea by building a software-specific domain lexical database based on tags on Stack Overflow and optimized the input queries to help search software-related documents. Chen et al. [78] reformulated the Chinese queries to English ones for searching related posts on Stack Overflow.

Most of these previous studies generate query reformulation based on heuristic rules or a lexical database, which depends greatly on the quality and size of the rules or the database. In contrast, SEQUER is fully data-driven (i.e., based on large-scale real-world developers' queries provided by Stack Overflow); we believe that SEQUER can automatically learn reformulation patterns, and generate better reformulation result.

C. Query reformulation for code search in SE

Many researchers study query reformulation for code search. Sisman et al. [30] proposed a query reformulation framework by enriching the users' queries with certain specific terms drawn from the highest-ranked retrieved artifacts. Howard et al. [79] leveraged similar word pairs in comments and method signatures, which are semantically similar in computer science situations to reformulate users' queries. Lu et al. [68] took a similar method, i.e., identifying each term in the original query but extends a query with synonyms generated from WordNet. Nie et al. [67] identified software-specific expansion words from high-quality pseudo relevance feedback question and answer pairs on Stack Overflow. Rahman et al. [80] identified terms from the source code using a novel term weight-CodeRank, and then suggested effective reformulation of the original query by exploiting the source document structures, query quality analysis, and machine learning. They further proposed a query reformulation technique that suggests a list of relevant API classes for a natural language query by exploiting keyword-API associations from the questions and answers on Stack Overflow [70]. Similarly, Sirres et al. [81] augmented the original query with structural code entities by mining questions and answers from Stack Overflow. Different from code search, our study mainly focuses on software-specific textual search. Moreover, SEQUER can complement these code search approaches to reformulate the users' queries better.

VII. Conclusion

Effective search among numerous questions and answers is a challenging task for developers, especially for novices. Our empirical study on a large number of real-world query records on Stack Overflow indicates that developers' queries are constantly reformulated to obtain the desired results. To assist with developers' efficient search, we propose a deep learning-based approach SEQUER to learn query reformulation patterns from query logs provided by Stack Overflow. Given the original query, it can automatically recommend a list of reformulation candidates for selection. Evaluation on large-scale archival query reformulations verifies the effectiveness of SEQUER compared with five state-of-the-art baselines.

In the future, we will further improve the performance of SEQUER by incorporating more contextual information such as the users' profile, query history, and their post visiting history. Moreover, we plan to take our approach one step further by directly recommending posts for the query. Based on users' queries and corresponding clicked posts provided by Stack Overflow, we could develop a domain-specific model to learn the relationships between them.

Acknowledgement

The authors would like to thank Stack Exchange Inc. for sharing the dataset, and the anonymous reviewers for their insightful comments and suggestions. This work is supported in part by the National Natural Science Foundation of China (Grant Nos. 61872263, 61702041 and 61202006), the Open Project of State Key Laboratory for Novel Software Technology at Nanjing University (Grant No. KFKT2019B14), and the Australian Research Council (DE180100153).

References

[1] Y. Huang, C. Chen, Z. Xing, T. Lin, and Y. Liu, "Tell them apart: distilling technology differences from crowd-scale comparison discussions," in *2018 33rd IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2018, pp. 214–224.

[2] C. Chen and Z. Xing, "Mining technology landscape from stack overflow," in *Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, 2016, pp. 1–10.

[3] C. Chen, Z. Xing, and L. Han, "Techland: Assisting technology landscape inquiries with insights from stack overflow," in *2016 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 2016, pp. 356–366.

[4] R. Abdalkareem, E. Shihab, and J. Rilling, "What do developers use the crowd for? a study using stack overflow," *IEEE Software*, vol. 34, no. 2, pp. 53–60, 2017.

[5] X. Xia, L. Bao, D. Lo, P. S. Kochhar, A. E. Hassan, and Z. Xing, "What do developers search for on the web?" *Empirical Software Engineering*, vol. 22, no. 6, pp. 3149–3185, 2017.

[6] C. Chen and Z. Xing, "Towards correlating search on google and asking on stack overflow," in *Proceedings of 2016 IEEE 40th Annual Computer Software and Applications Conference*. IEEE, 2016, pp. 83–92.

[7] R. Datta, D. Joshi, J. Li, and J. Z. Wang, "Image retrieval: Ideas, influences, and trends of the new age," *ACM Computing Surveys*, vol. 40, no. 2, pp. 1–60, 2008.

[8] Z.-J. Zha, L. Yang, T. Mei, M. Wang, Z. Wang, T.-S. Chua, and X.-S. Hua, "Visual query suggestion: Towards capturing user intent in internet image search," *ACM Transactions on Multimedia Computing, Communications, and Applications*, vol. 6, no. 3, pp. 1–19, 2010.

[9] C. Chen, Z. Xing, and X. Wang, "Unsupervised software-specific morphological forms inference from informal discussions," in *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)*. IEEE, 2017, pp. 450–461.

[10] X. Chen, C. Chen, D. Zhang, and Z. Xing, "Sethesaurus: Wordnet in software engineering," *IEEE Transactions on Software Engineering*, 2019.

[11] B. J. Jansen, D. L. Booth, and A. Spink, "Patterns of query reformulation during web searching," *Journal of the american society for information science and technology*, vol. 60, no. 7, pp. 1358–1371, 2009.

[12] M. Sloan, H. Yang, and J. Wang, "A term-based methodology for query reformulation understanding," *Information Retrieval Journal*, vol. 18, no. 2, pp. 145–165, 2015.

[13] L. Bing, W. Lam, T.-L. Wong, and S. Jameel, "Web query reformulation via joint modeling of latent topic dependency and term context," *ACM Transactions on Information Systems (TOIS)*, vol. 33, no. 2, pp. 1–38, 2015.

[14] J.-Y. Jiang, Y.-Y. Ke, P.-Y. Chien, and P.-J. Cheng, "Learning user reformulation behavior for query auto-completion," in *Proceedings of the 37th international ACM SIGIR conference on Research & development in information retrieval*, 2014, pp. 445–454.

[15] S. Y. Rieh *et al.*, "Analysis of multiple query reformulations on the web: The interactive information retrieval context," *Information Processing & Management*, vol. 42, no. 3, pp. 751–768, 2006.

[16] J. Huang and E. N. Efthimiadis, "Analyzing and evaluating query reformulation strategies in web search logs," in *Proceedings of the 18th ACM conference on Information and knowledge management*, 2009, pp. 77–86.

[17] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Proceedings of Advances in neural information processing systems*, 2014, pp. 3104–3112.

[18] A. Sordoni, Y. Bengio, H. Vahabi, C. Lioma, J. Grue Simonsen, and J.-Y. Nie, "A hierarchical recurrent encoder-decoder for generative context-aware query suggestion," in *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*, 2015, pp. 553–562.

[19] R. C. Cornea and N. B. Weininger, "Providing autocomplete suggestions," Feb. 4 2014, uS Patent 8,645,825.

[20] C. Sadowski, K. T. Stolee, and S. Elbaum, "How developers search for code: a case study," in *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, 2015, pp. 191–201.

[21] P. Mahdabi, M. Keikha, S. Gerani, M. Landoni, and F. Crestani, "Building queries for prior-art search," in *Proceedings of Information Retrieval Facility Conference*. Springer, 2011, pp. 3–15.

[22] D. Downey, S. Dumais, D. Liebling, and E. Horvitz, "Understanding the relationship between searchers' queries and information goals," in *Proceedings of the 17th ACM conference on Information and knowledge management*, 2008, pp. 449–458.

[23] R. Singh and N. S. Mangat, *Elements of survey sampling*. Springer Science & Business Media, 2013, vol. 15.

[24] A. J. Viera, J. M. Garrett *et al.*, "Understanding interobserver agreement: the kappa statistic," *Fam med*, vol. 37, no. 5, pp. 360–363, 2005.

[25] 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, pp. 1–22, 2018.

[26] C. Chen, Z. Xing, and Y. Liu, "By the community & for the community: a deep learning approach to assist collaborative editing in q&a sites," *Proceedings of the ACM on Human-Computer Interaction*, vol. 1, no. CSCW, pp. 1–21, 2017.

[27] J. Ooi, X. Ma, H. Qin, and S. C. Liew, "A survey of query expansion, query suggestion and query refinement techniques," in *Proceedings of 2015 4th International Conference on Software Engineering and Computer Systems*. IEEE, 2015, pp. 112–117.

[28] W. B. Croft, "Approaches to intelligent information retrieval." *Information Processing and Management*, vol. 23, no. 4, pp. 249–54, 1987.

[29] X. Wang and C. Zhai, "Mining term association patterns from search logs for effective query reformulation," in *Proceedings of the 17th ACM conference on Information and knowledge management*, 2008, pp. 479–488.

[30] B. Sisman and A. C. Kak, "Assisting code search with automatic query reformulation for bug localization," in *Proceedings of 2013 10th Working Conference on Mining Software Repositories*. IEEE, 2013, pp. 309–318.

[31] L. Bergroth, H. Hakonen, and T. Raita, "A survey of longest common subsequence algorithms," in *Proceedings Seventh International Symposium on String Processing and Information Retrieval. SPIRE 2000*. IEEE, 2000, pp. 39–48.

[32] T. Young, D. Hazarika, S. Poria, and E. Cambria, "Recent trends in deep learning based natural language processing," *IEEE Computational IntelligenCe Magazine*, vol. 13, no. 3, pp. 55–75, 2018.

[33] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Proceedings of Advances in neural information processing systems*, 2017, pp. 5998–6008.

[34] C. Liu, R. W. White, and S. Dumais, "Understanding web browsing behaviors through weibull analysis of dwell time," in *Proceedings of the 33rd international ACM SIGIR conference on Research and development in information retrieval*, 2010, pp. 379–386.

[35] K. Cao and M. Rei, "A joint model for word embedding and word morphology," *arXiv preprint arXiv:1606.02601*, 2016.

[36] R. Sennrich, B. Haddow, and A. Birch, "Neural machine translation of rare words with subword units," *arXiv preprint arXiv:1508.07909*, 2015.

[37] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *arXiv preprint arXiv:1409.0473*, 2014.

[38] J. Chen, C. Chen, Z. Xing, X. Xu, L. Zhu, G. Li, and J. Wang, "Unblind your apps: Predicting natural-language labels for mobile gui components by deep learning," in *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, ser. ICSE '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 322â€"334. [Online]. Available: https://doi.org/10.1145/3377811.3380327

[39] P. J. Liu, M. Saleh, E. Pot, B. Goodrich, R. Sepassi, L. Kaiser, and N. Shazeer, "Generating wikipedia by summarizing long sequences," *arXiv preprint arXiv:1801.10198*, 2018.

[40] N. Kitaev and D. Klein, "Constituency parsing with a self-attentive encoder," *arXiv preprint arXiv:1805.01052*, 2018.

[41] P. Koehn, "Pharaoh: a beam search decoder for phrase-based statistical machine translation models," in *Proceedings of the Conference of the Association for Machine Translation in the Americas*. Springer, 2004, pp. 115–124.

[42] A. Vaswani, S. Bengio, E. Brevdo, F. Chollet, A. N. Gomez, S. Gouws, L. Jones, L. Kaiser, N. Kalchbrenner, N. Parmar, R. Sepassi, N. Shazeer, and J. Uszkoreit, "Tensor2tensor for neural machine translation," *CoRR*, vol. abs/1803.07416, 2018.

[43] P. J. Werbos, "Backpropagation through time: what it does and how to do it," *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1550–1560, 1990.

[44] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[45] Q. Wang, B. Li, T. Xiao, J. Zhu, C. Li, D. F. Wong, and L. S. Chao, "Learning deep transformer models for machine translation," *arXiv preprint arXiv:1906.01787*, 2019.

[46] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[47] W. U. Ahmad, K.-W. Chang, and H. Wang, "Context attentive document ranking and query suggestion," in *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2019, pp. 385–394.

[48] J.-Y. Jiang and W. Wang, "Rin: reformulation inference network for context-aware query suggestion," in *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*, 2018, pp. 197–206.

[49] W. Chen, F. Cai, H. Chen, and M. de Rijke, "Attention-based hierarchical neural query suggestion," in *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*, 2018, pp. 1093–1096.

[50] W. U. Ahmad, K.-W. Chang, and H. Wang, "Multi-task learning for document ranking and query suggestion," in *International Conference on Learning Representations*, 2018.

[51] J. S. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl, "Algorithms for hyper-parameter optimization," in *Advances in neural information processing systems*, 2011, pp. 2546–2554.

[52] C. Napoles, K. Sakaguchi, M. Post, and J. Tetreault, "Ground truth for grammatical error correction metrics," in *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, 2015, pp. 588–593.

[53] ——, "Gleu without tuning," *arXiv preprint arXiv:1605.02592*, 2016.

[54] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu, "Bleu: a method for automatic evaluation of machine translation," in *Proceedings of the 40th annual meeting on association for computational linguistics*. Association for Computational Linguistics, 2002, pp. 311–318.

[55] D. Dahlmeier and H. T. Ng, "Better evaluation for grammatical error correction," in *Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2012, pp. 568–572.

[56] R. Oliveto, M. Gethers, D. Poshyvanyk, and A. De Lucia, "On the equivalence of information retrieval methods for automated traceability link recovery," in *Proceedings of 2010 IEEE 18th International Conference on Program Comprehension*. IEEE, 2010, pp. 68–71.

[57] C. McMillan, D. Poshyvanyk, and M. Revelle, "Combining textual and structural analysis of software artifacts for traceability link recovery," in *Proceedings of 2009 ICSE Workshop on Traceability in Emerging Forms of Software Engineering*. IEEE, 2009, pp. 41–48.

[58] G. Gay, S. Haiduc, A. Marcus, and T. Menzies, "On the use of relevance feedback in ir-based concept location," in *Proceedings of 2009 IEEE International Conference on Software Maintenance*. IEEE, 2009, pp. 351–360.

[59] B. Dit, M. Revelle, and D. Poshyvanyk, "Integrating information retrieval, execution and link analysis algorithms to improve feature location in software," *Empirical Software Engineering*, vol. 18, no. 2, pp. 277–309, 2013.

[60] C. Chen, Z. Xing, and Y. Liu, "What's spain's paris? mining analogical libraries from q&a discussions," *Empirical Software Engineering*, vol. 24, no. 3, pp. 1155–1194, 2019.

[61] C. Chen and Z. Xing, "Similartech: automatically recommend analogical libraries across different programming languages," in *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering*, 2016, pp. 834–839.

[62] C. Chen, Z. Xing, Y. Liu, and K. L. X. Ong, "Mining likely analogical apis across third-party libraries via large-scale unsupervised api semantics embedding," *IEEE Transactions on Software Engineering*, 2019.

[63] C. Chen, "Similarapi: mining analogical apis for library migration," in *2020 IEEE/ACM 42nd International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*. IEEE, 2020, pp. 37–40.

[64] J. Chen, C. Chen, Z. Xing, X. Xia, L. Zhu, J. Grundy, and J. Wang, "Wireframe-based ui design search through image autoencoder," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 29, no. 3, pp. 1–31, 2020.

[65] C. Chen, S. Feng, Z. Xing, L. Liu, S. Zhao, and J. Wang, "Gallery dc: Design search and knowledge discovery through auto-created gui component gallery," *Proceedings of the ACM on Human-Computer Interaction*, vol. 3, no. CSCW, pp. 1–22, 2019.

[66] C. Chen, S. Feng, Z. Liu, Z. Xing, and S. Zhao, "From lost to found: Discover missing ui design semantics through recovering missing tags," *Proceedings of the ACM on Human-Computer Interaction*, vol. 4, no. CSCW2, pp. 1–22, 2020.

[67] 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.

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

[69] M. M. Rahman, C. K. Roy, and D. Lo, "Automatic query reformulation for code search using crowdsourced knowledge," *Empirical Software Engineering*, vol. 24, no. 4, pp. 1869–1924, 2019.

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

[71] S. Haiduc and A. Marcus, "On the effect of the query in ir-based concept location," in *Proceedings of 2011 IEEE 19th International Conference on Program Comprehension*. IEEE, 2011, pp. 234–237.

[72] S. Haiduc, G. Bavota, R. Oliveto, A. Marcus, and A. De Lucia, "Evaluating the specificity of text retrieval queries to support software engineering tasks," in *Proceedings of 2012 34th International Conference on Software Engineering*. IEEE, 2012, pp. 1273–1276.

[73] S. Haiduc, G. Bavota, R. Oliveto, A. De Lucia, and A. Marcus, "Automatic query performance assessment during the retrieval of software artifacts," in *Proceedings of the 27th IEEE/ACM international conference on Automated Software Engineering*, 2012, pp. 90–99.

[74] 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 2013 35th International Conference on Software Engineering*. IEEE, 2013, pp. 842–851.

[75] S. Haiduc, G. De Rosa, G. Bavota, R. Oliveto, A. De Lucia, and A. Marcus, "Query quality prediction and reformulation for source code search: The refoqus tool," in *Proceedings of 2013 35th International Conference on Software Engineering*. IEEE, 2013, pp. 1307–1310.

[76] M. M. Rahman and C. K. Roy, "Quickar: automatic query reformulation for concept location using crowdsourced knowledge," in *Proceedings of 2016 31st IEEE/ACM International Conference on Automated Software Engineering*. IEEE, 2016, pp. 220–225.

[77] Z. Li, T. Wang, Y. Zhang, Y. Zhan, and G. Yin, "Query reformulation by leveraging crowd wisdom for scenario-based software search," in *Proceedings of the 8th Asia-Pacific Symposium on Internetware*, 2016, pp. 36–44.

[78] G. Chen, C. Chen, Z. Xing, and B. Xu, "Learning a dual-language vector space for domain-specific cross-lingual question retrieval," in *2016 31st IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2016, pp. 744–755.

[79] M. J. Howard, S. Gupta, L. Pollock, and K. Vijay-Shanker, "Automatically mining software-based, semantically-similar words from comment-code mappings," in *Proceedings of 2013 10th Working Conference on Mining Software Repositories*. IEEE, 2013, pp. 377–386.

[80] M. M. Rahman and C. K. Roy, "Improved query reformulation for concept location using coderank and document structures," in *Proceedings of 2017 32nd IEEE/ACM International Conference on Automated Software Engineering*. IEEE, 2017, pp. 428–439.

[81] R. Sirres, T. F. Bissyandé, D. Kim, D. Lo, J. Klein, K. Kim, and Y. Le Traon, "Augmenting and structuring user queries to support efficient free-form code search," *Empirical Software Engineering*, vol. 23, no. 5, pp. 2622–2654, 2018.