BIKER: A Tool for Bi-Information Source Based API Method Recommendation

Liang Cai, Haoye Wang, Qiao Huang
Zhejiang University
China

Xin Xia
Monash University
Australia

Zhenchang Xing
Australian National University
Australia

David Lo
Singapore Management University
Singapore

ABSTRACT

Application Programming Interfaces (APIs) in software libraries play an important role in modern software development. Although most libraries provide API documentation as a reference, developers may find it difficult to directly search for appropriate APIs in documentation using the natural language description of the programming tasks. We call such phenomenon as knowledge gap, which refers to the fact that API documentation mainly describes API functionality and structure but lacks other types of information like concepts and purposes. In this paper, we propose a Java API recommendation tool named BIKER (Bi-Information source based KnowledgE Recommendation) to bridge the knowledge gap. We implement BIKER as a search engine website. Given a query in natural language, instead of directly searching API documentation, BIKER first searches for similar API-related questions on Stack Overflow to extract candidate APIs. Then, BIKER ranks them by considering the query's similarity with both Stack Overflow posts and API documentation. Finally, to help developers better understand why each API is recommended and how to use them in practice, BIKER summarizes and presents supplementary information (e.g., API description, code examples in Stack Overflow posts) for each recommended API. Our quantitative evaluation and user study demonstrate that BIKER can help developers find appropriate APIs more efficiently and precisely.

Demo Tool Website: http://biker.net.cn/

Demo Video: https://youtu.be/BVu29JIAuXY

CCS CONCEPTS

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

KEYWORDS

API Recommendation, API Documentation, Stack Overflow

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

ACM ISBN 978-1-4503-5572-8/19/08…$15.00
https://doi.org/10.1145/3338906.3341174

ACM Reference Format:
Liang Cai, Haoye Wang, Qiao Huang, Xin Xia, Zhenchang Xing, and David Lo. 2019. BIKER: A Tool for Bi-Information Source Based API Method Recommendation. In *Proceedings of the 27th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE '19), August 26–30, 2019, Tallinn, Estonia.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3338906.3341174

1 INTRODUCTION

In modern software development, the Application Programming Interfaces (APIs) provided by software libraries bring great convenience to software development. However, it is not easy to be familiar with all APIs in a large library. For example, for Java SE 8 API, there are more than 4K classes and 30K methods. Thus, developers often need to check the API documentation to learn how to use an unfamiliar API. However, a common scenario is that developers only have the requirement of a programming task, while they do not even know which API is worth learning.

A possible solution for this problem is to use the natural language description of the programming task as a query, and use Information Retrieval (IR) approaches to obtain some candidate APIs whose documentation is *semantically similar* to the query. However, our preliminary experiment has shown that the solution may fail to find the relevant API if its description does not share semantically similar words with the query. We call such mismatches between a task description and the API documentation as *task-API knowledge gap*, and our observation is also consistent with previous studies (e.g., [10]), which pointed out that API documentation mainly describes API functionality and structure, but lacks other types of information (e.g., concepts or purposes).

In our previous work [7], to bridge this task-API knowledge gap, we conducted a survey with 130 Java developers to understand how developers search for APIs to resolve programming tasks. We found that, instead of directly checking API documentation, most developers would choose first to browse several relevant Stack Overflow (SO) questions and pick out the APIs that seem to be useful in the discussions. Inspired by this information seeking process, we proposed an automatic approach named BIKER (Bi-Information source based KnowledgE Recommendation) which leverages both SO posts and API documentation to recommend APIs for a programming task.

In this paper, we strengthen BIKER by implementing it as a publicly accessible search engine website. Using BIKER, developers can use natural language to describe the Java programming task as a query. Given a query, BIKER will output the top-5 Java API methods

that are most likely to solve the task. To help developers better understand why these APIs are recommended so that they can make their decisions more easily, BIKER also provides supplementary information (i.e., official API description, relevant SO questions and code snippets) for each recommended API.

To evaluate BIKER, we compare it with two baselines (i.e., RACK [8] and DeepAPI [6]) using our manually labeled dataset and the dataset published by RACK. The results show that our tool significantly outperforms RACK and DeepAPI by at least 42% in terms of MAP and MRR. We also conduct a user study in which 28 Java developers are divided into four groups using different tools to answer 10 Java-API-method-related questions randomly sampled from SO. The results show that our tool can help developers find the correct APIs for Java programming tasks more efficiently and accurately. Finally, we release the replication package of BIKER [2] to help other researchers replicate and extend our work.

2 APPROACH

Figure 1 shows the overall framework of BIKER, which consists of three main components: building domain-specific language models for similarity calculation (Section 2.1), searching for relevant APIs based on SO posts and API documentation (Section 2.2), and summarizing API supplementary information (Section 2.3).

2.1 Building Language Models for Similarity Calculation

During the search process of BIKER, a key step is to calculate the semantic similarity between two pieces of texts. To measure a query's similarity to a SO post or an API description, we need to build domain-specific language models. We first build a text corpus by extracting the text content from SO posts that are tagged with *java*. We remove long code snippets enclosed in HTML tag ⟨*pre*⟩, but keep short code fragments between ⟨*code*⟩ tags as natural language sentences. We use NLTK [4] to tokenize the sentences. Using the SO corpus, we train a word embedding model using Gensim [9]. Word embedding model provides the basic model to measure word similarity. Then we build the word IDF (inverse document frequency) vocabulary. A word's IDF represents the inverse of the number of SO posts that contain the word. We reduce each word in the corpus to its root form (aka. stemming) using NLTK. Thus, the words with the same root form will have the same IDF value. The more posts in which a word appears, the less likely the word carries important semantic information, and thus its IDF is lower. We use IDF as a weight on top of word embedding similarity.

2.2 Searching for Relevant APIs

Our API search component performs three steps: retrieve similar SO questions to the query, detect API entities in the SO posts, and calculate the query's similarity with SO posts and API descriptions for ranking the relevance of candidate APIs to the query.

2.2.1 Retrieving Similar Questions. Given a query describing a programming task, the first step is to retrieve the top-k similar questions from SO. BIKER first transforms the text of a question's title and the query into two bags of words, denoted as T and Q, respectively. Then an asymmetric similarity score from T to Q is computed as a normalized, IDF-weighted sum of similarities between words in T and all words in Q:

$$sim(T \rightarrow Q) = \frac{\sum_{w \in T} sim(w, Q) \times idf(w)}{\sum_{w \in T} idf(w)} \quad (1)$$

where $sim(w, Q)$ is the maximum value of $sim(w, w')$ for each word $w' \in Q$, and $sim(w, w')$ is the cosine similarity of the word embedding vectors of w and w'. The asymmetric similarity score $sim(Q \rightarrow T)$ is computed analogously, by swapping T and Q in Equation 1. Intuitively, a word with lower IDF value would contribute less to the similarity score. Finally, the similarity score between T and Q is computed as the *harmonic mean* of the two asymmetric scores:

$$sim(T, Q) = \frac{2 \times sim(T \rightarrow Q) \times sim(Q \rightarrow T)}{sim(T \rightarrow Q) + sim(Q \rightarrow T)} \quad (2)$$

The retrieved top-k similar questions will be used to detect candidate APIs for recommendation. In this paper, BIKER only retrieves the top-50 similar questions, since retrieving too many questions may introduce noise to the recommendation process.

2.2.2 Detecting API Entities. After retrieving the top-k similar questions, BIKER uses several heuristic rules to extract API entities from each question's answers. These APIs are considered as candidate APIs for recommendation. If an API is not mentioned in any of the top-k similar questions, it is less likely to be the right API for the query. Thus, we do not consider all APIs of a language or library for recommendation. In this way, a lot of irrelevant APIs would be filtered out.

To detect API entities, we first manually checked a large number of API-related questions. We observe that an important API mentioned by developers is often highlighted with the HTML tag ⟨*code*⟩ or referenced by a hyperlink to the API's corresponding documentation page. Thus, BIKER detects API entities using the following two heuristics:

- BIKER checks every hyperlink in each answer and uses regular expressions to identify the hyperlink to a Java official API documentation site. Then it uses regular expressions to detect the full name of the corresponding API method from the hyperlink and mark this method as a candidate API.
- BIKER builds a dictionary that stores the names of all Java APIs. Then it checks the text contained in every HTML tag ⟨*code*⟩ in each answer. If the text fully matches any API method in the dictionary, it is marked as a candidate API.

2.2.3 Calculating Similarity Score for Ranking Candidate APIs. After obtaining a list of candidate APIs from the top-k similar questions, BIKER calculates the similarity score between each candidate API and the query. Given an API and a query Q, their similarity score is a combination of two scores, namely *SimSO* and *SimDoc*. Specifically, *SimSO* measures the similarity between the query and the question title T of a top-k similar question in which the API is mentioned, and *SimDoc* measures the similarity between the query and the API's description in the official API documentation.

Suppose that among all the top-k similar questions, the API is mentioned in n questions, then *SimSO* is computed as:

$$SimSO(API, Q) = min(1, \frac{\sum_{i=1}^{n} sim(T_i, Q)}{n} \times \log_2 n) \quad (3)$$

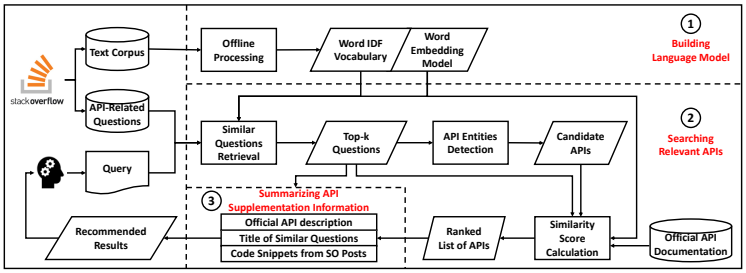

Figure 1: Overall framework of BIKER

where $sim(T_i, Q)$ represents the similarity score between the query and the title of the i-th question that mentions the API, and $sim(T_i, Q)$ is calculated based on Equation 2. *SimSO* considers two aspects. First, the score should be related to the similarity between each question and the query. Thus, it calculates the average of the similarity score between each question's title and the query. Second, if the API is mentioned in multiple questions, it is more likely to be the right API for the query. Thus, the score is further boosted based on the number of questions. We add a logarithm transformation $\log_2 n$ to control the scale of boosting. For example, the score would be boosted by 20% if the API is detected in 4 questions. We also restrict that the boosted score should not exceed 1.

The *SimDoc* is also calculated based on Equation 2 given the query Q and the API description D. Finally, the similarity score between the query and the API is the harmonic mean of the corresponding *SimSO* and *SimDoc*.

2.3 Summarizing API Supplementary Information

After obtaining the ranked list of candidate APIs, BIKER summarizes supplementary information for each API in the list. We do this following the findings of our developer survey (see [7]) which suggest that developers usually need to check more information about API description and API usage examples to decide which API should be chosen for their tasks. Thus, the supplementary information summarized by BIKER considers three aspects, including the official API description, the title of similar questions and the code snippets that contain the corresponding API.

3 TOOL IMPLEMENTATION AND USAGE

We implement BIKER in the form of a search engine website. The following subsections describe BIKER's knowledge base, implementation and usage.

3.1 Knowledge Base

We extracted 1,347,908 questions tagged with Java from the official data dump [3] of SO (published on: Dec 9th, 2017). Then we built a text corpus based on the text content in these questions and their answers to train a word embedding model and construct an IDF vocabulary with the text corpus. To create the knowledge base of API-related questions for similar questions retrieval, we select the questions with positive score and at least one positive-scored answer to the question contains API entities. In this way, we collected 125,847 API-related questions. Finally, we downloaded the Java SE 8 API documentation [1] and parsed the HTML file of each API class to extract all API methods, along with their descriptions.

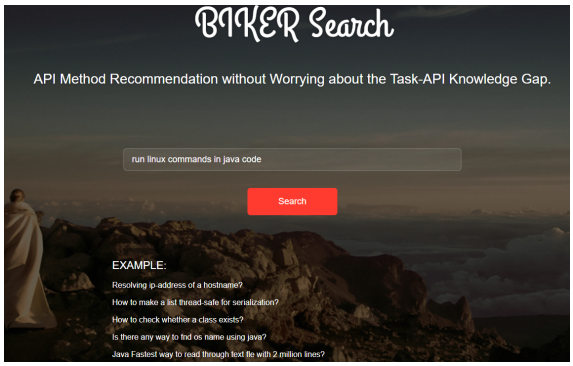

Figure 2: The homepage of BIKER

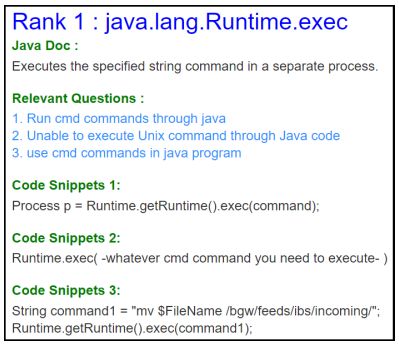

Figure 3: The top-1 recommendation result for the query "run linux commands in java code"

In total, we extracted 31,736 methods and built a dictionary mapping the name of each method with its description in the documentation.

3.2 Tool Implementation

Figure 2 shows the homepage of BIKER. When a developer sends a textual query to the server, the query will be processed in the background. By default, BIKER returns the top-5 APIs that are most likely to solve the task, along with supplementary information for each API. For example, given the query "*run linux commands in java code*", Figure 3 shows the top-1 recommended API and its supplementary information. Among these pieces of information, BIKER first presents the official API description so that developers can quickly check the API's functionality. Then, BIKER presents the title of similar SO questions whose answers mentioned this API. These questions are ranked by their titles' similarity scores with the query in descending order. In case a developer is interested to

further investigate the discussion in a SO question, the developer can click the title which is a hyperlink to the corresponding SO webpage of the question. To reduce information overload, at most three similar questions would be presented. Finally, BIKER checks each similar question's answers and extracts code snippets containing the API. Specifically, given an API, a code snippet is extracted if it satisfies both the following conditions: 1) The number of lines of code is no more than five; 2) The API's class name and method name are both contained in the code snippet. To reduce information overload, it presents at most three code snippets. Thus, developers can check these code snippets to understand how to use the API.

3.3 Usage Scenarios

In this section, we present several examples to illustrate how developers would interact with BIKER. For queries like "*how to get the length of a string*", developers can quickly find the correct answer (i.e., *String.length*) when browsing the name and document description of the top-1 recommended API. However, in many cases, it is not easy to quickly figure out whether the recommended API is useful by only checking its name and document description. For example, given the query "*How to free memory in Java*", although the top-1 recommended API (i.e., *System.gc*) is the right answer, its document description (i.e., "*Runs the garbage collector*") is not semantically overlapping with the query. The developer may not understand the API usage if he/she is not familiar with Java's garbage collector. However, since all of the three related SO questions are asking similar topics like how to free Java object or heap, the developer can click one of these questions and he/she will learn the mechanism of Java's memory management by reading the posts.

In some cases, the code snippets recommended by BIKER may help developers further understand the API usage and make it easier to apply the API to developers' own code. For example, given the query "*How to round a number to n decimal places in Java*", the top-1 recommended API (i.e., *Math.round*) is the right choice. For this API, BIKER recommended code snippets for rounding a number to 2 decimal places, which can be easily extended to n decimal places with a simple modification.

One drawback of BIKER is that it only recommends individual API, while some programming tasks may require an API sequence. To find the appropriate sequence, developers can leverage the code snippets provided by BIKER. For example, given the query "*Remove trailing zeros from double*", the top-1 recommended API (i.e., *BigDecimal.stripTrailingZeroes*) seems to be the right answer since its name is straightforward and the key phrase in its documentation (i.e., *with any trailing zeros removed*) also seems to meet the task requirement. However, this API would transform a number like 600.0 into a scientific notation, which may cause some bugs when directly printing the result. To fix this issue, developers need to call *BigDecimal.toPlainString* after stripping the trailing zeros, which can be easily found in BIKER's recommended code snippets.

Note that sometimes developers may not be able to clearly describe the query. For example, when a developer wants to know the Java API for sorting, if he/she inputs "*sorting algorithm*", BIKER cannot output relevant APIs since the query is too short and the purpose is not that specific. A better query could be "*sort a list*" or "*sorting algorithm java api*". In future work, we plan to improve BIKER by guiding developers to clarify their queries.

4 EVALUATION

4.1 Quantitative Evaluation

To evaluate the effectiveness of BIKER, we manually selected 413 Java-API-related questions from SO and labeled the ground-truth APIs for these questions based on their accepted answers. We compare BIKER with two baselines (i.e., RACK [8] and DeepAPI [6]) using our dataset and the dataset published by RACK. The results show that our tool significantly outperforms RACK and DeepAPI by at least 42% in terms of MAP and MRR. Readers can check our previous work [7] for more details about the evaluation results.

To evaluate the efficiency of BIKER, we record its query processing time. On average, BIKER takes 2.8 seconds to process each query. The major computation cost for query processing is due to the step of similar questions retrieval, where BIKER needs to compare the query with the titles of about 120 thousand questions. To improve the time efficiency, we can reduce the size of questions to be compared with some heuristic rules (e.g., only comparing with the question whose vote score is larger than k) or accelerate similarity score computation (i.e., matrix multiplication) by parallel computation using GPU [5].

4.2 User Study

To further investigate how developers interact with BIKER and whether it can help developers find correct APIs more efficiently and accurately, we conducted a user study with 28 participants from both university and IT companies. All of them have Java development experience in either commercial or open source projects, with an average of 2.9 years of development experience. These participants are divided into four groups using different tools to answer 10 Java-API-related questions randomly sampled from our testing dataset. Readers can check our previous work [7] for more details about the sampled questions and group settings. On average, compared with the other three groups (i.e., web search only, using DeepAPI and using BIKER with only API name recommendation but no supplementary information), the group using the full version of BIKER can improve answer correctness by 11%-23% and save answering time by 28%-49%.

5 CONCLUSION AND FUTURE WORK

In this paper, we present BIKER, a tool implemented as a search engine website to automatically recommend relevant APIs for a programming task described in natural language. Inspired by the information seeking process of developers, we leverage both Stack Overflow posts and API documentation to improve the effectiveness of BIKER, and summarize supplementary information for each recommended API to help developers better understand the API usage and determine their relevance to the query task. In future work, we plan to extend BIKER to support more programming languages (e.g., Python) and provide more customized search options.

ACKNOWLEDGMENTS

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

REFERENCES

[1] 2017. Java SE 8 API documentation downloading site. http://www.oracle.com/technetwork/java/javase/documentation/jdk8-doc-downloads-2133158.html.

[2] 2017. The replication package of BIKER. https://github.com/tkdsheep/BIKER-ASE2018.

[3] 2017. Stack Overflow Data Dump. https://archive.org/download/stackexchange.

[4] Steven Bird and Edward Loper. 2004. NLTK: the natural language toolkit. In *Proceedings of the ACL 2004 on Interactive poster and demonstration sessions*. Association for Computational Linguistics, 31.

[5] Kayvon Fatahalian, Jeremy Sugerman, and Pat Hanrahan. 2004. Understanding the efficiency of GPU algorithms for matrix-matrix multiplication. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*. ACM, 133–137.

[6] Xiaodong Gu, Hongyu Zhang, Dongmei Zhang, and Sunghun Kim. 2016. Deep API learning. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*. ACM, 631–642.

[7] Qiao Huang, Xin Xia, Zhenchang Xing, David Lo, and Xinyu Wang. 2018. 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, 293–304.

[8] Mohammad Masudur Rahman, Chanchal K Roy, and David Lo. 2016. Rack: Automatic api recommendation using crowdsourced knowledge. In *Software Analysis, Evolution, and Reengineering (SANER), 2016 IEEE 23rd International Conference on*, Vol. 1. IEEE, 349–359.

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

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