A Fusion of Java Domain Knowledge Base and Siamese Network for Java API Recommendation

Hao Li  
Department of Software  
Yunnan University  
Kunming Yunnan 650500 China  
lihao707@ynu.edu.cn

Tao Li  
Department of Software  
Yunnan University  
Kunming Yunnan 650500 China  
365318663@qq.com

Sheng Zhong  
Department of Software  
Yunnan University  
Kunming Yunnan 650500 China  
542293454@qq.com

Yan Kang (Corresponding author)  
Department of Software  
Yunnan University  
Kunming Yunnan 650500 China  
kangyan@ynu.edu.cn

Tie Chen  
Department of Software  
Yunnan University  
Kunming Yunnan 650500 China  
962870113@qq.com

Abstract—APIs play an important role in modern software development. Programmers need to frequently search for the appropriate APIs according to different tasks. With the development of the information industry, API reference documents have become larger and larger. Due to redundant and erroneous information on the Internet, traditional search methods can also cause inconvenience to programmers’ queries. At the same time, there is a gap in terms of vocabulary and knowledge between the natural language description of the programming task and the description in the API documentation, so it is difficult to find a suitable API. To solve these problems, this paper proposes a Java API recommendation model by fusing the Java domain knowledge base and the Siamese Network to improve the accuracy of API recommendation. Experiments on the BIKER data set show that our method has better recommendation results than the state-of-art DeepAPI and BIKER model.

Keywords—API recommendation, deep learning, Java, Stack Overflow, BERT

I. INTRODUCTION

In software development, APIs (Application Programming Interfaces) provided by software library brings great convenience for software development. However, it is not easy to be familiar with all the APIs in a large software library. For example, when Java 1.0 was released in 1996, its JDK (Java Development Kit) contained 211 classes and interfaces. When Java 8.0 was released in 2014, its JDK contained 4240 classes and interfaces [1], and more than 130000 methods [2]. In 18 years, the number of classes and interfaces increased 20 times. In the actual development process, in addition to the use of standard libraries, programmers will also use a large number of third-party libraries. In this situation, programmers will generally use search engines or consult official API documents to locate the required APIs.

But these two methods have their own shortcomings. At present, search engines are basically general search engines, and API recommendation is a domain-closed requirement. This kind of conflict results from many irrelevant or even wrong information in the process of using search engine query APIs. Using the method of querying the API documentations to find the APIs can obtain highly relevant and high-quality results, but because the API documentations is usually large, it will consume a lot of time and effort in the query process. In addition, since the solution of a problem is usually completed by a combination of multiple APIs, relying only on official API documents will make the whole development process very inefficient [3].

In order to solve the above problems, many code recommendation systems have been developed. Different from general search engines, code recommendation systems are mainly to solve the problem of code search. In the code recommendation systems, some systems recommend code segment and some systems recommend APIs. This paper mainly studies API recommendation system.

At present, there are two kinds of API recommendation systems: information retrieval model and deep learning model. At the beginning, API recommendation systems were dominated by information retrieval model. It mainly includes APIBot [4], RACK [5], RUSH [6], BIKER [7], etc. The technologies they use mainly include TF-IDF [10], bag-of-words [9], etc. In recent years, with the rapid development of deep learning, many new technologies and models are proposed, which are used in API recommendation systems. There are API recommendation systems based on deep learning, mainly to include DeepAPI [8] [19], CODEnn [11], NLP2Code [12], etc. The core algorithm they use is RNN (Recurrent Neural Network) [13]. All the above systems have achieved good results in the field of API recommendation, but there are still some problems. First, the accuracy of API recommendation is insufficient, and second, the existing domain knowledge base is not better used to solve the new API query question.
In view of the above problems, this paper proposes the FOSAR (Fusion of Java Domain Knowledge Base and Siamese Network for Java API Recommendation) model. The main contributions of this model are as follows:

1) We propose a new method for extracting APIs from the adopted answers in Stack Overflow. This method can reduce the noise extracted from the answer. In the first step, the method splits the adopted answer text. In the second step, the method uses the class/interface names to match all the words obtained in the previous step. If the match is successful, the class is considered as a candidate class/interface. Finally, all the method names of the candidate classes are matched with all the words obtained in the first step. If the match is successful, the method-level API formed by the method and its class/interface is used as candidate API.

2) We propose a new matching method for unknown and known questions, which is different from the most popular methods based on Word2Vec [17] and bag-of-words. Firstly, we use the sentence vector generated by BERT to calculate the semantic similarity between the new question and the known Stack Overflow questions, and screen out 100 known Stack Overflow questions closest to the new question. Secondly, we use Siamese Network [20] with BERT [7] to recalculate the similarity score between the new question and the 100 Stack Overflow questions. Thirdly, the 100 Stack Overflow questions are sorted by recalculated similarity scores, and we can get the s most similar Stack Overflow questions. s is the number of the most similar questions we finally choose, it is a natural number greater than 0 and less than 100. We call these s Stack Overflow questions as top-s Stack Overflow questions. Finally, we find out all the APIs in the top-s Stack Overflow questions, and get the recommended list according to the similarity between the sentence vector of new question and the sentence vector of the API description.

II. RELATED WORK
A. Direct recommendation model based on API documents
Most of the early API recommendation systems used traditional information retrieval technology to directly match questions and API documents. Taking APIBot [4] as an example, it mainly uses traditional methods such as TF-IDF [10] and regular expression to match questions and documents. But because API documents mainly describe an API function, most of the questions raised by programmers are purpose descriptions, that is, simply describe the effect to be achieved. Such as “how do I generate random integers within a specific range in Java?”, but description of java.util.Random.nextInt is “Returns a pseudorandom, uniformly distributed int value between 0 (inclusive) and the specified value (exclusive), drawn from this random number generator's sequence.” This kind of problem and document matching are difficult to achieve a high degree of matching, so researchers began to try to use other matching methods instead of directly questions and documents matching.

B. Indirect recommendation model based on API documents
Due to the inherent differences between the questions and the API documents, researchers began to find an intermediate quantity to make up for the differences between the questions and the API documents. There is typical RACK [5] and RUSH [6]. The main purpose is to match the questions raised by the programmers and the questions have been solved in Stack Overflow, and then return the matching successful APIs. This method has achieved good results. But there are two difficulties which have great influence on the results. The first difficulty is the matching between the unknown questions and the known questions. Now the most popular methods are based on Word2Vec [17], such as [23], which transforms each question into a vector to match. This method can solve this kind of problem very well, because there is a high degree of matching for similar words. But for deeper semantics, there is difficult for matching, which is an urgent problem to be solved.

The second difficulty is to extract the APIs in the known questions. Now there are two main methods to extract the APIs in the known questions. One is to directly match the APIs in the answers. The advantage of this method is that it can basically extract the correct APIs. The disadvantage is that there will be a lot of noise in the extracted API, because few of the answers only mention the correct API, and there will be many other APIs adulterated among them. These noises of APIs will greatly affect the recommendation result. The other is to use regular expression to match the hyperlink [6] to Java document in the answer. The advantage of this method is that the extracted APIs is very pure and noise free. The disadvantage is that most of the answers do not contain hyperlinks to Java API documents, so the constructed knowledge base will be very small, which will affect the final effect.

C. API recommendation model based on deep learning
There has not been a relatively authoritative and large data set in the API recommendation field, which leads to the failure of neural network training. Thus, deep learning technology was not applied in API recommendation until 2016. DeepAPI [8] collected all 442928 Java projects with more than 1 stars from 2008 to 2014 on GitHub. The system takes the first sentence of the annotation in the code as the question, and parses the corresponding code back to AST (Abstract Syntax Tree) and extracts the corresponding APIs, forming the {question: APIs} pairs. The characteristic of this type of system is to improve the accuracy by using a large amount of data. The disadvantage is similar to the recommendation systems directly based on API documents, because the code annotation still describes the functionality of an API, which does not match the questions of programmers very well, but because of its huge amount of data, its accuracy is improved compared with the recommendation systems directly based on API documents.

III. OUR FOSAR MODEL
The FOSAR proposed in this paper is mainly used for Java API recommendation. It reads a new API question input by programmers, and then outputs the corresponding APIs after system processing. The model proposed in this paper is divided into two parts. The first part is to build a Java domain knowledge base, which will be used in the future similarity analysis. The second part is to generate APIs recommended list. The architecture of FOSAR model is shown in Fig. 1.

A. The establishment of Java domain knowledge base
In this layer, we build Java domain knowledge base according to Java API documents and Stack Overflow.
community. This layer consists of two steps. The first step is to extract knowledge from the Java SE 8 API document [16]. In the second step, we extract knowledge from the crawled Stack Overflow web page. Finally, all extracted knowledge forms Java domain knowledge base.

1) Extracting knowledge from Java API documents
In this part, we downloaded the Java SE 8 API document and wrote a python script to automatically extract the APIs from the html files of the document. During the extraction process, first, the classes/interfaces and their functions form the {class or interface: methods, full name} pairs, such as {'ArrayList': ['ensureCapacity', 'add', 'clear'...], 'java.util.ArrayList'}. Methods inherited from the parent classes/interfaces are also included in the method list. Next, we mainly study method-level API recommendation, so the functions of each class and the class serve as an API. The corresponding function descriptions and API form the {API: descriptions} pairs, such as {'java.io.CharArrayReader.read': ['Reads a single character.', 'Reads characters into a portion of an array.']}. The above API is described in two ways because the read method has two overloaded forms.

All of {class or interface: methods, full name} pairs and all of the {API: descriptions} pairs are the extracted knowledge.

2) Extracting knowledge from Stack Overflow
Stack Overflow is a comprehensive technology community with various types of IT-related (Information Technology) questions. In this system, we only extract Java-related questions. We have written a crawler automatic crawling questions and answers in Stack Overflow. First of all, through the tags on Stack Overflow, we can filter out the questions with Java tags. The quality of some questions is not high, mainly because the descriptions of the questions is unclear or there is no answer. None of the above problems can be used by the system, so we need to further filter the questions with Java tag on Stack Overflow. We only extract questions with the scores higher than one, and the questions have adopted answers.

Next, we wrote a python script to automatically extract APIs from the adopted answers. The purpose is that after extracting the API in the corresponding answer of each question, it can be processed once and used multiple times. Compared with the current mainstream BIKER model, this method avoids the repeated extraction of the APIs contained in the same question, and improves the time efficiency of the system to a certain extent.

We found that if the text of the adopted answer is matched with the class name or method name, it will introduce a lot of noise. For example: if the answer contains "List.addItem", java.awt.List.add and java.awt.List.addItem will be extracted.
The generation of the API recommended list

In this layer, we use the previously established Java domain knowledge base and deep learning methods to generate API recommendation list. In the first step, the BERT semantic similarity scores of a new question and the known Stack Overflow questions are calculated, and the 100 most similar Stack Overflow questions can be obtained by using similarity score sorting. The second step is to recalculate the similarity scores of the new question and the 100 Stack Overflow questions through fusing BERT with the Siamese Network. We sort the 100 Stack Overflow questions by the recalculated similarity scores, and find the top-s Stack Overflow questions. Finally, we sort the APIs that exist in the answer to the top-s Stack Overflow questions to get the recommended list.

The obtention of the 100 most similar Stack Overflow questions

API recommendation systems have the problem of less data, so it is very necessary to mine useful data from dimensionless data. In natural language processing, one of the most direct and effective tasks to use dimensionless data is language model, so many tasks use language model as pre-training task.

BERT [14] is not the first unsupervised language model. Compared with other language models (such as ELMo [15], which has only three layers), the advantage of BERT is depth model and unsupervised model. The depth of BERT is reflected in that it uses transformer [18] as decoder. Transformer has achieved great success in machine translation task, and can do it very deeply. The pre-training model used in this paper has 24 layers and more than 300 million parameters. BERT’s unsupervised training is reflected in its training process. BERT uses the data of 3.3 billion words from books corpus and English Wikipedia to train the BERT model.

In this step, in order to save FOSAR prediction time in the next step. We first use the BERT vectors to filter out the 100 Stack Overflow questions that are most similar to the new question. The reason for choosing the closest 100 Stack Overflow questions is that in the comparison of 637881 Stack Overflow questions, FOSAR can save the time to recalculate the score in fusing with the Siamese Network in next step. In fact, the value of 100 can be selected as any natural number according to the number of APIs that need to select.

When a natural language is input into the BERT operation, we obtain the output of layer 23 as the semantics of the natural language. The reason for choosing layer 23 is that the output of the last layer is too close to the result of the original training task, which will reduce the semantic accuracy. After BERT processing, natural language $n$ can be transformed into sentence vector. As shown in (1).

$$v_n = \text{BERT}(S_n)$$ (1)

Where $S_n$ is the natural language description of natural language $n$.

BERT can transform the title of each Stack Overflow question and a new API question into sentence vector. Next, we need to calculate the similarity between the sentence vector of the new API question and the sentence vector of each known Stack Overflow question. The similarity between the known API question $m$ and the new API question $n$ can be described by (2).

$$bscore_{mn} = \frac{v_m \cdot v_n}{|v_m|}$$ (2)

Where $v_m$ and $v_n$ represent the sentence vector of the known API question $m$ and the sentence vector of the new API question $n$ respectively. $|v_n|$ is the Modulus of the $v_n$. This formula is similar to the cosine similar function, but it does not contain Modulus of the $v_n$. The reason is that when matching for new questions, the modules will not change, will not affect the overall ranking of scores, and reduce the amount of calculation.

After obtaining the similarity scores of all known Stack Overflow questions and new API questions, we can sort the Stack Overflow questions according to the similarity score to obtain the 100 most similar Stack Overflow questions.
data in IV.D shows that the introduction of the Siamese Network strategy can indeed improve the accuracy of the final recommendation. This Siamese Network is also used to calculate the similarity between two questions. The architecture of Siamese Network is shown in the Fig. 2.

![Siamese Network Diagram](image)

**Fig. 2.** The architecture of Siamese Network

In order to alleviate the large amount of calculation problems caused by dimension, we set the dimension of our Word2Vec is 50, and all sentences in SNLI (The Stanford Natural Language Inference) data set are used as corpus to train the Word2Vec model. The trained Word2Vec is used to generate word vectors for fixed words.

The generation process of word vector for a word \( w_t \) can be obtained by (3).

\[
c_t = \text{Word2Vec}(w_t)
\]

(3)

In the first layer of Siamese Network, the two sentences are embedded by the trained Word2Vec model. For two question \( (S_i, S_j) \) input into this layer, its embedding process first obtains the vector of each word in each sentence according to Word2Vec. Then, all the word vectors of each sentence are spliced to form there present embedding. The process can be described as (4) and (5).

\[
e_j = [e_{j1}, e_{j2}, ..., e_{jk}]
\]

(4)

\[
e_k = [e_{k1}, e_{k2}, ..., e_{kp}]
\]

(5)

Where \( e_j \) and \( e_k \) represent embedding of question \( S_j \) and question \( S_k \) respectively. \( 0 \) and \( P \) represent the length of questions \( S_j \) and \( S_k \) respectively. \( e_{jt} \) represents the \( t \)-th word of sentence \( S_j \).

In the second layer of Siamese Network, the embedding \( e_j \) and \( e_k \) are put into the same BiLSTM (Bidirectional Long and Short Term Memory) and obtain the vectors of two sentences respectively. The process can be described as (6) and (7).

\[
h_k = \text{lstm}(e_k) \oplus \text{lstm}(e_j)
\]

(6)

\[
h_j = \text{lstm}(e_j) \oplus \text{lstm}(e_j)
\]

(7)

Where \( \text{lstm}(e_j) \) and \( \text{lstm}(e_k) \) represent the result of LSTM sequence iteration and reverse iteration respectively, and \( \oplus \) represents the operation of vector connection.

In the third layer of Siamese Network, the language vectors of two sentences are concatenated, and then the final similarity score is obtained by sigmoid function after passing through the full connection layer. The process can be described as (8).

\[
sscore_{jk} = \frac{1}{1 + e^{-(h_k \oplus h_j)W + b}}
\]

(8)

Where \( W \) and \( b \) represent the parameter matrix and bias vector of the full connection layer respectively.

We trained the Siamese Network with SNLI data set. Finally, the similarity scores of the 100 Stack Overflow questions and the new question are recalculated according to the similarity of Siamese Network and BERT. For the known API question \( m \) and the new API question \( n \), the final formula for recalculating the similarity score is obtained by (9).

\[
score_{mn} = \lambda \cdot bscore_{mn} + (1 - \lambda) \cdot sscore_{mn}
\]

(9)

Where \( \lambda \) is a hyper-parameter and \( \lambda \in (0,1) \).

After recalculating the similarity scores, we reorder 100 Stack Overflow questions according to the new similarity scores, and then we can find out the most similar \( s \) Stack Overflow questions. We call the \( s \) Stack Overflow questions as top-\( s \) questions.

3) Generation of recommendation list

In this step, we need to sort the APIs in the \( s \) questions. The similarity score of API descriptions and the new question is calculated to determine the order of APIs. The algorithm is shown in Algorithm 1.

**Algorithm 1**

**Input:** the top-\( s \) Stack Overflow questions \( top_s \) questions, all known \{API: descriptions\} pairs \( api_{\text{descriptions}} \), the BERT vector of the new question \( v_n \).

**Output:** recommendation list RL.

1. for pair \( \in m_s \) questions do:
2. \( \text{apis} = \text{pair.getAPIs()} \)
3. for api \( \in \text{apis} \) do:
4. \( \text{descriptions} = \text{api.descriptions.get_descriptions(api)} \)
5. \( d_{\text{vs}} = \text{BERT(descriptions)} \)
6. for \( v_n \in d_{\text{vs}} \) do:
7. \( \text{bscore}_{mn} = \text{avg} \left( \frac{v_n \cdot v_m}{|v_n|} \right) \)
8. \( \text{bscore}_{mn} = \text{avg} \left( \frac{v_n \cdot v_m}{|v_n|} \right) \)
9. RL.append((API: \text{bscore}_{mn}))
10. sort( RL)
where sort method first sorts the APIs in top-1 Stack Overflow questions by bscore value. Then, all APIs in the remaining s-1 Stack Overflow questions are then sorted according to bscore values. Finally, the two sorting results are connected to form a recommendation list.

IV. EXPERIMENTAL AND ANALYSIS RESULTS

A. Dataset

In this experiment, the data set published by BIKER is used as the test, and the data set contain 413 function-level Q & A pairs. All of the 413 function-level Q not in our Java domain knowledge base.

B. Evaluation indicator

In this experiment, we use Hit@K, MRR@K and MAP@K to evaluate our model and compare them with two advanced models: DeepAPI and BIKER model. MRR and MAP are also widely used in previous software engineering studies [21-22]. In these three indicators, Hit@K measures the correctness of the API recommended by the systems, while MRR@K and MAP@K measure the quality of the API sequence recommended by the model. In the API recommendation systems, the correctness of the recommended API is very important, but apart from giving the correct recommendation, the order of recommendation is also very important. If the correct API can be put in the front as much as possible, it can save programmer time and improve work efficiency to a large extent. Hit@K measures the proportion of questions that can be answered correctly to the total number of questions when only K APIs are recommended. The formula is (10).

\[
\text{Hit@K} = \frac{1}{n} \sum_{i=1}^{n} \frac{1}{\sum_{j=1}^{T} \text{rel}(R_j, T_i)}
\]  

(10)

where n represents the total number of test questions, \( R_j \) represents the j-th recommended API, \( T_i \) represents the known APIs of the i-th list, \( T \) represents the number of APIs in \( T_i \). And \( \text{rel}(R_j, T_i) \) is a binary function that returns 1 when the recommended API at position j is in \( T_i \), otherwise return 0.

MRR@K measures the order of the recommended results. The higher the positions of the correct answers in the recommended results are, the better the results are. The formula of MRR@K is (11).

\[
\text{MRR@K} = \frac{1}{n} \sum_{i=1}^{n} \sum_{j=1}^{T} \text{bscore}(j, R_i, T_i)
\]  

(11)

Where, The function of \( \text{bscore}(j, R_i, T_i) \) return 1/j when \( R_i \) in \( T_i \), otherwise return 0.

MAP@K also measures the order of recommendation results, which is also a very important indicator in recommendation systems. MAP also considers the order of correct answers in recommendation results. The formula is (12).

\[
\text{MAP@K} = \frac{1}{n} \sum_{i=1}^{n} \frac{1}{\sum_{j=1}^{T} \text{bscore}(j, R_i, T_i) \cdot \text{posd}(j, R_i, T_i)}
\]  

(12)

C. Model setting

The BIKER model uses Word2vec and TF-IDF to recommend API. FOSAR-OB (Only BERT) is a simplified FOSAR model that only uses BERT to calculate similarity. The Hit@5 value cannot be measured because the DeepAPI online system fails in [19]. It can be seen from the TABLE I that the quality of our model’s recommended results under the data set of BIKER is improved evidently under MRR@5, MAP@5 and Hit@5 indicators compared with DeepAPI and BIKER. Our model improved 32.4%, 26.8% and 24.23% respectively in MRR@5, MAP@5 and Hit@5 evaluation indicators.

### TABLE I. RECOMMENDED QUALITY AT METHOD-LEVEL UNDER BIKER DATA SET

<table>
<thead>
<tr>
<th>methods</th>
<th>DeepAPI</th>
<th>BIKER</th>
<th>FOSAR-OB</th>
<th>FOSAR</th>
<th>Improvement</th>
</tr>
</thead>
<tbody>
<tr>
<td>MRR@5</td>
<td>0.188</td>
<td>0.6031*</td>
<td>0.7656</td>
<td>0.7985</td>
<td>32.4%</td>
</tr>
<tr>
<td>MAP@5</td>
<td>0.153</td>
<td>0.5018*</td>
<td>0.6222</td>
<td>0.6363</td>
<td>26.8%</td>
</tr>
<tr>
<td>Hit@5</td>
<td>---</td>
<td>0.6896*</td>
<td>0.8298</td>
<td>0.8567</td>
<td>24.23%</td>
</tr>
</tbody>
</table>

![Fig. 3. Influence of λ value on evaluation indicators](image-url)
In TABLE I, we found that the recommendation effect of FOSAR-OB using only BERT does not have a higher accuracy than FOSAR. It is proved that recalculation of similar score in section III.B.2 can improve the accuracy of recommendation. Fig. 3. shows the influence of $\lambda$ value on evaluation indicators. It can be found that the best result is when $\lambda$ is set to 0.7. When the value of $\lambda$ is set to 1, the result is not the best. It is also proved that recalculation of similar score in section III.B.2 can improve the accuracy of recommendation. TABLE II shows a comprehensive comparison of BIKER and FOSAR on some specific questions.

### TABLE II. A COMPREHENSIVE COMPARISON OF BIKER AND FOSAR ON SOME SPECIFIC QUESTIONS

<table>
<thead>
<tr>
<th>QID</th>
<th>Query</th>
<th>Answers</th>
<th>BIKER Recommendation</th>
<th>FOSAR Recommendation</th>
<th>B_MAP@5</th>
<th>A_MAP@5</th>
</tr>
</thead>
<tbody>
<tr>
<td>146</td>
<td>How do you kill a thread in java?</td>
<td>java.lang.Thread.stop</td>
<td>java.lang.Thread.stop</td>
<td></td>
<td>0.25</td>
<td>0.5</td>
</tr>
<tr>
<td>30</td>
<td>How to split a comma-separated string?</td>
<td>java.util.Arrays.asList</td>
<td>java.util.Arrays.asList</td>
<td></td>
<td>0.25</td>
<td>1.0</td>
</tr>
<tr>
<td>60</td>
<td>How to get all possible values of an enum in java? (not knowing the specific Enum)</td>
<td>java.lang.Class.getEnumConstants</td>
<td>java.lang.Class.getEnumConstants</td>
<td></td>
<td>0.5</td>
<td>1.0</td>
</tr>
<tr>
<td>309</td>
<td>Check if class exists without running into ClassNotFound Exception</td>
<td>java.lang.Class.forName</td>
<td>java.lang.Class.forName</td>
<td></td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>289</td>
<td>Setting log level for inner class (in Glassfish)</td>
<td>java.util.logging.Logger.getLogger</td>
<td>java.util.logging.Logger.getLogger</td>
<td></td>
<td>0.25</td>
<td>0.5</td>
</tr>
<tr>
<td>145</td>
<td>Java Round up Any Number</td>
<td>java.lang.Math.ceil</td>
<td>java.lang.Math.ceil</td>
<td></td>
<td>0.5</td>
<td>0.25</td>
</tr>
</tbody>
</table>

In TABLE I, we found that the recommendation effect of FOSAR-OB using only BERT does not have a higher accuracy than FOSAR. It is proved that recalculation of similar score in section III.B.2 can improve the accuracy of recommendation. Moreover, the recommended API cannot get accurate parameters of each function, but function overloading is common in the object-oriented program language. It is also our future work.
Acknowledgment

This work has been supported by the National Science Foundation of China Grant No. 61762092, “Dynamic multi-objective requirement optimization based on transfer learning”, No. 61762089, “The key research of high order tensor decomposition in distributed environment”, and the Open Foundation of the Key Laboratory in Software Engineering of Yunnan Province, Grant No. 2017SE204,” Research on extracting software feature models using transfer learning”.

REFERENCES