Prompt Enhance API Recommendation: Visualize the User's Real Intention behind this Query

Yong Wang[1,2†], Linjun Chen[1*], Cuiyun Gao[3†], Yingtao Fang[1†], Yong Li[4†]

[1]School of computer and information, Anhui Polytechnic University, Jiujiang District, Wuhu, 241000, Anhui, China.
[2]Anhui Artificial Intelligence Laboratory, Institute of Artificial Intelligence, Hefei Comprehensive National Science center, High-tech Zone, Hefei, 230026, Anhui, China.
[3]School of Computer Science and Technology, Harbin Institute of T echnology, , Shenzhen, 518055, Guangdong, China.
[4]College of Computer Science and Technology, Xinjiang Normal University, , Urumqi, 830017, Xinjiang, China.

*Corresponding author(s). E-mail(s): 2210920113@stu.ahpu.edu.cn;
Contributing authors: yongwang@ahpu.edu.cn; gaocuiyun@hit.edu.cn;
2269721052@qq.com; liyong@live.com;
†These authors contributed equally to this work.

Abstract

Developers frequently rely on APIs in their daily programming tasks, as APIs have become an indispensable tool for program development. However, with a vast number of open-source libraries available, selecting the appropriate API quickly can be a common challenge for programmers. Previous research on API recommendation primarily focused on designing better approaches to interpret user input. However, in practical applications, it is often difficult for users, especially novice programmers, to express their real intentions due to the limitations of language expression and programming capabilities.

To address this issue, this paper introduces PTAPI, an approach that visualizes the user's real intentions based on their query to enhance recommendation performance. Firstly, PTAPI identifies the prompt template from Stack Overflow (SO) posts based on the user's input. Secondly, the obtained prompt template is combined with the user's input to generate a new question. Finally, the newly

generated question leverages dual information sources from SO posts and API official documentation to provide recommendations.
To evaluate the effectiveness of PTAPI, we conducted experiments at both the class-level and method-level. The experimental results demonstrate the effectiveness of the proposed approach, with a significant improvement in the success rate.

Keywords: API Recommendation, Real Intentions, Prompt Template, Stack Overflow

1 Introduction

In the process of modern software development, because the application program interface (API) can improve the quality of code and improve the development efficiency of the program, it is widely used by developers. Researchers randomly selected 1008 open-source projects on GitHub for research, and found that 93.3 % of the projects used third-party libraries, and an average of 28 third-party libraries were invoked per project [1]. Obviously, API plays a very important role in the software development process. Through research, it is found that the number of APIs is very large and appears very rapidly [2, 3]. In the past 20 years, the number of Java Development Kit (JDK) APIs has increased by more than 20 times, from 211 in the first edition of 1996 to 4403 in 2022 [4, 5]. The emergence of new APIs is accompanied by the failure of old APIs. It is impossible for developers to understand all APIs. During the development process, developers often need to select these unfamiliar APIs. In the process of learning and then using these APIs, they usually consult the API reference document. However, through investigation, it is found that these API documents often have problems such as content redundancy, incompleteness, inaccurate description, ambiguity and lack of examples [6]. To learn and use these APIs, programmers need to spend a lot of time. According to research, developers need to spend 40 % of their time learning APIs during the development process [7].

To solve this problem, relevant researchers have proposed API recommendation approaches. According to current research on API recommendation, we divide API recommendation into two types: completion type, which involves recommending subsequent related content based on the existing code context, and question-and-answer type, which allows developers to obtain answers through questions they enter. For the completion type, MAPO [8] and UPMiner [9] use frequent patterns or clustering techniques to mine the API usage patterns obtained in the project. PAM [10] uses the probability statistical model of API call sequence to obtain the usage pattern of API. Recent studies such as FOCUS [11], GAPI [12], MEGA [13], etc., mainly based on the current code context of developers, use collaborative filtering technology to calculate similarity and make subsequent recommendations. For question-and-answer, most of them return the final answer by entering the function they want to achieve on different question answering websites. For example, RACK [14] builds a database and then queries whether the keyword is mapped to the answer. BIKER [15] uses the similarity between the calculated input question and the posts and official documents in SO to

obtain the final answer. BRAID [16] adds feedback function on this basis. PICASO regards the query as a code annotation, looks for positive and negative samples in the post in SO, and forms a triple to form multiple input sources [17]. After studying the question-and-answer model, we found a major problem.

The objective of the aforementioned approach is to bridge the knowledge and vocabulary gap that exists between the natural language description of programming tasks and API-related documents. The knowledge gap refers to the lack of important task description information, such as purpose and concept, in API documentation, which primarily focuses on the structure and function of the API. The vocabulary gap arises from the potential presence of multiple lexical expressions conveying the same semantics. These approaches address the practical problem faced by users by continuously incorporating diverse information sources. However, it is important to acknowledge that the problems raised by users often involve uncertainty. We find that this phenomenon is particularly common among novice programmers. Their uncertainty often stems from inconsistencies between their queries and search terms, or from difficulties in accurately expressing their true intentions.

In this paper, we propose a task-based user intent visualization approach PTAPI(use **P**rompt **T**emplate to enhance **API** recommendation performance). This approach introduces SO as a third-party information source, and uses similar post problems in SO as a prompt template for task description, so that developers can understand their real intentions. That is, the developer input the problem to be solved, find out the most similar problems from SO, and the developer selects the most similar problem as the prompt template, and combines the two sentences into a new input problem. Then, the new questions are sent to the query, and an API recommendation list is obtained by extracting the API answers in SO to match the similarity with the official documents. Finally, the newly obtained API list is reordered and fed back to the user.

To validate the effectiveness of PTAPI, we utilized problem posts from Stack Overflow (SO) as our dataset and assessed the model's validity at both the method and class levels. In order to evaluate the efficacy of prompt learning in API recommendation, we examined the selection and placement of prompt templates. To gather user feedback on the model, we enlisted the participation of several graduate students and undergraduates who provided their experiences with using PTAPI, resulting in predominantly positive evaluations. Furthermore, the experimental results demonstrate the superiority of our approach compared to the baseline method.

In summary, our main contributions in this paper include :

1) We propose a new approach framework, PTAPI, to bridge the gap between user description and actual intention.

2) We propose to construct a prompt template for the user 's input to visualize the user 's real intention.

3) We conduct extensive experiments, and the results show that our proposed approach is effective.

The rest of this paper is organized as follows. The Section 2 describes the motivation example of this paper. The Section 3 elaborates our approach. The Section 4 introduces the evaluation settings, and the Section 5 introduces the experimental

results. The Section 6 discusses our work, and the Section 7 introduces the related work. Finally, the Section 8 summarizes this paper.

2 MOTIVATION EXAMPLE

In this section, we will introduce the motivation for writing this paper and illustrate the importance of prompts for recommendations.

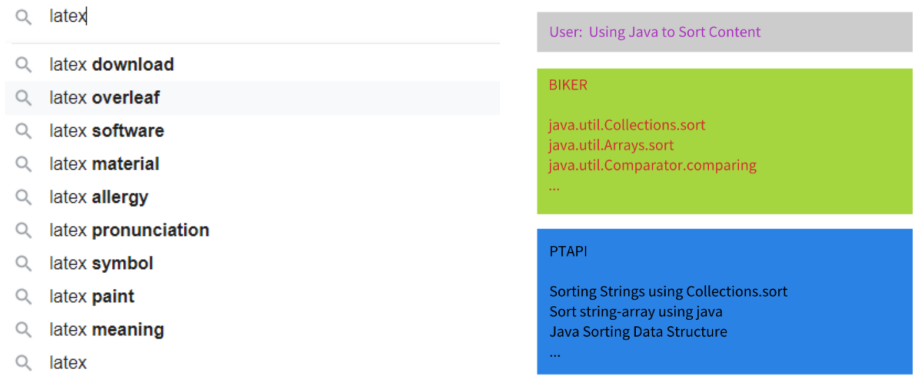
By observing the question posts in SO, we find that a large number of questions are ambiguous, and the questioners do not know how to express their needs, so that the respondents cannot provide help. After investigation, we found that the language description of user problems is not only related to the user 's language expression ability. Moreover, it is also closely related to the user 's programming experience. In the process of program development, developers often encounter problems, but they often cannot express them clearly. At present, researchers have proposed API recommendation to solve these situations. They always consider the solution from the downstream, that is, constantly designing better approaches to bridge the user 's description, but they ignore the user 's true intention.

By using browsers, shopping and other software that support user input, we found that these software are very friendly to new users. They not only have a good use of instructions, but also, when the user searches in the search box, the input of different content will have different prompt statements to guide. As shown in Figure 1 (a), when the user searches for latex in the browser, the latex-related prompts appear automatically below. Many of these prompt statements are search records from themselves or other users. According to this discovery, we think about whether this prompt approach can be used in API recommendation, so we propose a approach to add a prompt template for API recommendation, so as to visualize the user 's description and better bridge the vocabulary gap with the questions in the question and answer library. Combined with the method of prompt learning, when the user enters the problem, the model returns several similar problems. Users can select expression statements similar to their own descriptions, so that they can search more accurately. As shown in Figure 1 (b) , when the user wants to sort a set of data, he enters ' Using Java to Sort Content '. For other model approaches, they will give a lot of APIs with sorting function. If the user is not familiar with the given results, it is necessary for the user to search further. For users, they prefer that when they enter the above questions, there is a prompt telling them to sort the collection, array, or other data. Specifically, we first create a high-quality problem library to generate a prompt template to display the user 's implicit purpose. Finally, the appropriate API recommendation is given according to the corresponding template prompt. Compared with the original recommendation idea, our recommendation idea has been changed as shown in Figure 2.

3 METHODOLOGY

3.1 Workflow of PTAPI

Figure 3 illustrates the workflow of our framework. PTAPI mainly includes three main stages : constructing a language model for subsequent similarity calculation,



(a) Example of recommendation sequences from browser

(b) Example of API recommendation sequences from PTAPI and BIKER

Fig. 1: Example diagram

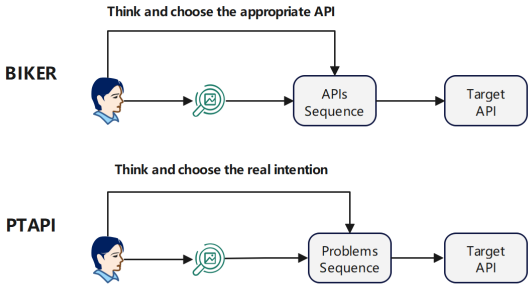


Fig. 2: Recommendationl idea diagram

generating a prompt template based on the input problem and generating new problems, continuing to search for new problems and generating APIs that need to be recommended.

3.2 Training language model

In this section, we will build a corpus of training models to measure the similarity between queries and SO posts or API documents. The data in this corpus is extracted from the SO website, which contains a large number of SO posts on API issues. We extract the text information of the posts from the HTML page and process it. On the one hand, we delete posts with no answers and unclear answers. On the other hand, we also deleted the long code fragments in the post, because these long code fragments will affect the performance of the training and the results of the recommendation.

For the bag-of-words based word embedding model, we choose the Word2Vec [18] model in a neural network to train. Word2Vec model is a kind of language model,

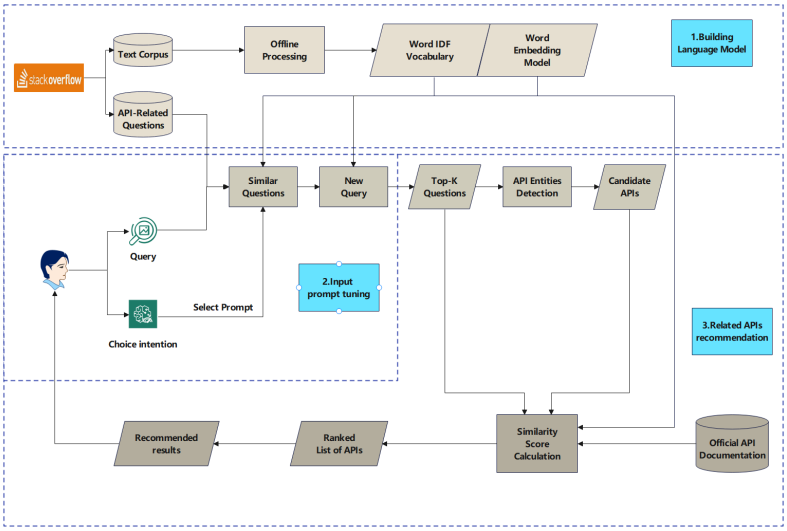


Fig. 3: Overall framework of PTAPI

which can learn semantic knowledge from a large number of text predictions in an unsupervised way. At present, it can be widely used in natural language processing, and the trained word embedding model can measure the similarity between words. Then we construct an IDF (Inverse Document Frequency) vocabulary, which can reflect the frequency of words in all texts. If a word appears in many texts, its IDF value will be lower and its importance will be lower. We use the NLTK [19] library to preprocess the text of the corpus and normalize the words in the corpus, that is, convert each word into a root form, and the IDF value of the words with the same root is the same. We use IDF to weight the word embedding similarity. Finally, the trained IDF vocabulary and word embedding are also used directly for the words in the API document, because the vocabulary in the SO post is much larger than the vocabulary in the API official document.

3.3 Generate prompt template

At this section, we need to generate prompt through the input description, and the newly generated prompt statements can better express the user 's display information. First, we need to find the most similar questions to the input description in the SO post title. Through the language model constructed in the previous stage, we calculate the similarity formula 1 between the query and the SO problem as follows :

$$sim(\mathrm{W}_q, \mathrm{W}_{SO}) = cos(\mathrm{W}_q, \mathrm{W}_{SO}) = \frac{\mathrm{W}_q^T \mathrm{W}_{SO}}{\|\mathrm{W}_q\| \, \|\mathrm{W}_{SO}\|} \tag{1}$$

W_q and W_{SO} represent the words in Q_q and Q_{SO} respectively. We use cosine similarity to calculate similarity. The cosine similarity is normalized by calculating the inner product of two vectors and using the L2 norm. To calculate the similarity between sentences, we first use the formula 2 to measure the similarity between sentences.

$$sim(W_q, Q_{SO}) = \max_{W_{SO} \subseteq Q_{SO}} (W_q, Q_{SO}) \tag{2}$$

Then, we need to calculate the maximum similarity between the word and each word in the sentence, and use the IDF vocabulary to weight it, as shown in the following formula 3, 4:

$$sim(Q_q, \overset{sim}{\rightarrow} Q_{SO}) = \frac{\sum_{W_q \subseteq Q_q} sim(W_q, Q_{SO}) * IDF(W_q)}{\sum_{W_q \subseteq Q_q} IDF(W_q)} \tag{3}$$

$$sim(Q_{SO}, \overset{sim}{\rightarrow} Q_q) = \frac{\sum_{W_{SO} \subseteq Q_q} sim(W_{SO}, Q_q) * IDF(W_{SO})}{\sum_{W_{SO} \subseteq Q_{SO}} IDF(W_{SO})} \tag{4}$$

$IDF(W_{SO})$ and $IDF(W_q)$ represent the IDF weights of the words W_{SO} and W_q, convert the Q_{SO} and Q_q into two word bundles, and then calculate the similarity scores from Q_q to Q_{SO}, and from Q_{SO} to Q_q, respectively. Finally, the two scores are combined to calculate the final similarity score, as shown in the formula 5 :

$$sim(Q_{SO}, Q_q) = \frac{1}{\frac{1}{sim(Q_q \overset{sim}{\rightarrow} Q_{SO})} + \frac{1}{sim(Q_{SO} \overset{sim}{\rightarrow} Q_q)}} \tag{5}$$

Using the above formula, the similarity between the user 's input and the statement in SO can be calculated, and finally several similar statements are returned to the user. According to the returned statement, the user selects the statement that meets his own purpose as the prompt template.

3.4 Generate relevant API recommendations

In the previous section, we find the prompt template according to the description entered by the user, and we combine the generated template with the description entered by the user into a new input statement. The operation that continues the previous stage calculates the sentence similarity for the newly generated statement. By calculating, n similar questions are retrieved in SO posts, and there are several answers in these similar questions. Therefore, in the next chapter, we study the setting of n at the class level and the method level respectively. Too much may introduce noise, too little may ignore the correct answer.

Through the previous step, we get several answers. Because the answers in the SO post contain many hyperlinks of the API official document, it is necessary to use the regularized expression to extract the hyperlinks contained in the HTML tag. And the regularization expression can extract the names of API methods in hyperlinks, we mark these APIs as candidate APIs.

After the previous step, we get several candidate API entities. There may be different solutions to a problem, especially at the class level. Therefore, we need to further sort these API entities. Specifically, we calculate the similarity score between the newly generated query and the post title containing the candidate API answer. We calculate the similarity score between the newly generated query and the API description in the official API document, as shown in Formula 6. Through these two scores, the final API answer ranking is obtained.

$$sim_{so} = min(1, \frac{\sum_{i=1}^{n} sim(Q_{so}, Q'_q)}{n}) * (1 + \log_2 n) \quad (6)$$

where $Q'_q = Q_q + max(sim(Q_{so}, Q_q))$ represents the newly generated problem description, sim_{so} represents the problem description and the post title with SO, and sim_{doc} represents the problem description and the official API document. The final score is shown in Formula 7.

$$Sim_{total} = \frac{2 * Sim_{so} * Sim_{doc}}{Sim_{so} + Sim_{doc}} \quad (7)$$

4 EXPERIMENTAL SETUP

In this section, we evaluate the proposed approach by answering three research questions :

•**RQ1** : What effect does our approach have on the experimental results ?

•**RQ2** : What is the effect of selecting different numbers of SO posts as the source of API sequence on the experimental results ?

•**RQ3** : What is the effect of the combination sequence and combination times of the prompt template on the experimental results ?

4.1 Dataset Description

To verify the effectiveness of the prompt template approach, we selected the BIKER dataset for experiments. BIKER 's dataset is extracted from SO. SO is an open Q & A website, and the quality of many posts in it is not particularly high. Hence, the BIKER researchers conducted rigorous screening procedures. For the dataset employed in prompt template generation and API answer search, the screening criteria were as follows: 1) the question must have a positive score, and 2) the question must possess at least one answer, with the answer also having a positive score. Consequently, a dataset comprising 125,847 questions was formed after the screening process. Additionally, a separate testing dataset consisting of 413 datasets was created. This testing dataset comprised high-quality questions and ground-truth APIs. Specifically, the SO problem score had to be equal to or greater than 5 points, and these data were excluded from the initial dataset. The test data involved inputting query questions and comparing them with their respective answers, ultimately evaluating the accuracy of the experiment.

4.2 Baselines

Baseline1(**BIKER**) [15]: uses a bag-of-words based word embedding model to narrow the vocabulary gap and knowledge gap by using SO posts and API official documents. BIKER implements recommendation at class level and method level.

Baseline2(**BRAID**) [16]: reorders the recommended results by using the user 's (implicit) feedback information. It belongs to a framework that combines experiments with BIKER, RACK, and NLP2API approaches, and it performs best on BIKER.

4.3 Evaluation Metrics

To evaluate the effectiveness of PTAPI, we compare the two indicators of MRR (Mean Reciprocal Rank) and MAP (Mean Average Precision) with the baseline, which are common evaluation indicators in the field of information retrieval and software engineering [20–23]. MRR indicates how far it takes to find the first correct answer in the recommendation list. MAP represents the ranking of the correct answers in the result ranking. In addition, we also use S @ K to represent the first correct API position in the top K of the recommendation list. The calculation formula of each evaluation index is as follows :

$$MRR = \frac{\sum_{R}^{i=1} \frac{1}{pos_i}}{R} \tag{8}$$

where R denotes the number of all queries, and pos_i denotes the number of ranked digits of the first answer to the i-th question.

$$MAP = \frac{1}{R} \sum_{i=1}^{R} \frac{\sum_{i=1}^{n} count(hit_i) * rel(i)}{len(true\_apis)} \tag{9}$$

where $count(hit_i)$ denotes the number of correct APIs in the preceding i. $rel(i)$ denotes whether the given sentence with the order of i is the true answer sentence, 1 denotes yes, and 0 denotes no.

$$S@K = \frac{\sum_{i=1}^{R} count(rank_i <= K)}{R} \tag{10}$$

where $rank_i <= K$ indicates that the position of the first correct answer is within K.

4.4 Implementation Details

We run our experiments on a computer with AMD Ryzen 7 2700X 3.7GHz, 32GB RAM. We found that retaining similar questions as prompt templates has a great effect on recommendation. Too few prompts may miss the goals that users need, and too many prompts may cause users to choose. Finally, we retain the top 10 most similar problems as templates. After subsequent experiments, it is found that due to the high quality of the test data set problem, the first result is always the best. For the selection of the number of recommended results, we conducted a special study in the next chapter. To compare with the baseline, we selected the parameter of 50 for the experiment.

5 Results

5.1 Effectiveness of PTAPI Compared with Baselines (RQ1)

Table 1 shows the overall results of all baselines and PTAPI in terms of S @ 1, S @ 3, S @ 5, MRR and MAP metrics.

Comparison at the Method Level. Compared with BIKER, we have increased by 27.2 %, 20.3 %, 11.9 %, 14.3 %, and 18.1 % on S @ 1, S @ 3, S @ 5, MAP, and MRR, respectively. Compared with BRAID, we increased the S @ 1, S @ 3, S @ 5, MAP, MRR by 22.3 %, 18.3 %, 11.6 %, 11.9 %, 15.7 %, respectively.

Table 1: Comparison of PTAPI and baselines results

	method level					class level				
	S@1	**S@3**	**S@5**	**MAP**	**MRR**	**S@1**	**S@3**	**S@5**	**MAP**	**MRR**
BIKER	0.423	0.660	0.775	0.553	0.569	0.547	0.814	0.903	0.675	0.652
BRAID	0.440	0.671	0.780	0.565	0.581	0.562	0.817	0.905	0.689	0.703
PTAPI	**0.538**	**0.794**	**0.867**	**0.632**	**0.672**	**0.697**	**0.889**	**0.932**	**0.761**	**0.794**

Comparison at the Class Level. Compared with BIKER, we have increased by 22.3 %, 18.3 %, 11.6 %, 11.9 %, and 15.7 % on S @ 1, S @ 3, S @ 5, MAP, and MRR, respectively. Compared with BRAID, we increased the S @ 1, S @ 3, S @ 5, MAP, MRR by 24.0 %, 8.8 %, 3.0 %, 10.4 %, 12.9 %, respectively.

It is worth noting that PTAPI has the most obvious increase on S @ 1 at both the method level and the class level. This means that PTAPI can recommend more than half of the correct APIs in the S @ 1 results. This is consistent with our expectation that the prompt template can help users better choose the API. BRAID implicitly expresses the user 's possible willingness by using the user 's feedback information, and we display the user 's willingness through the prompt template. The results show that the display improvement effect is better.

Prompt template is very important for API recommendation, which can significantly improve the recommendation performance.

5.2 The influence of parameters on the experiment (RQ2)

Table 2 and Table 3 are the different parameter recommendation results of PTAPI at the method level and the class level, respectively.

Table 2: At the method-level, various performance comparisons are made when the number of different similar problems is taken

	S@1	**S@3**	**S@10**	**MAP**	**MRR**
5	0.550	0.666	0.685	0.573	0.609
15	0.557	0.782	0.792	0.613	0.654
20	**0.566**	0.746	0.816	0.624	0.666
30	0.561	0.782	0.864	0.633	0.679
40	0.552	0.804	0.898	**0.641**	**0.682**
50	0.538	0.794	0.906	0.632	0.672
100	0.527	**0.809**	0.935	0.629	0.667
1000	0.462	0.814	**0.973**	0.571	0.619

Table 3: At the class-level, various performance comparisons are made when the number of different similar problems is taken

	S@1	**S@3**	**S@10**	**MAP**	**MRR**
30	0.690	0.872	0.922	0.748	0.781
40	0.692	0.881	0.939	0.754	0.788
45	**0.702**	0.884	0.944	**0.763**	**0.796**
50	0.697	0.889	0.947	0.761	0.794
100	0.685	**0.896**	0.964	0.763	0.789
1000	0.617	0.884	**0.998**	0.718	0.747

The parameter here refers to the number of similar SO posts retained in the recommendation results, and the final API ranking is obtained in these similar SO posts. To study this problem, we manually adjusted and set multiple parameters for experiments. By analyzing the experimental results in the two tables, we find that when the parameter setting is very small, the accuracy of the experiment is relatively poor. At the method level, when the parameter is 20, S @ 1 works best. On the overall trend, S @ 3 and S @ 10 increase with the increase of parameters. Among them, when the experimental parameter is 40, the MAP and MRR are optimal. At the class level, when the experimental parameter is 45, the results of S @ 1, MAP and MRR are the best.

From the analysis of the above results, we know that when the number of similar problems is too small, the experiment may lose important API information. When the number of reservations is too large, a lot of noise will be introduced. S @ 3 and S @ 10 have a good fault-tolerant space (such as S @ 10 indicates the probability of a correct answer in the recommended 10 answers), and S @ 3 and S @ 10 will also decrease as the number of parameters continues to increase. Fortunately, users pay more attention to the top-ranked answers. Therefore, we pay more attention to the results of S @ 1, MAP and MRR in the experiment.

The experimental results show that the number of similar problems is not the larger or the smaller the better, it is necessary to analyze the specific problems.

5.3 The influence of the position of the prompt template on the experiment (RQ3)

Table 4 is the experimental results of the prompt template at different positions, and table 5 is the experimental results of the existence of multiple prompt templates.

Table 4 shows the experimental results at the method level (at the class level, the sequence of the prompt template does not affect the experimental results), where K indicates that the prompt template is in front and the user 's input problem is in the back. $\hat{K}$ indicates that the user 's input problem is ahead and the prompt template is behind. We found that regardless of the number of parameters, the result of putting

Table 4: The influence of the position of the template on the experiment was suggested

	S@1	**S@3**	**S@10**	**MAP**	**MRR**
30	**0.562**	**0.782**	**0.864**	**0.633**	**0.679**
3$\hat{0}$	0.540	0.760	0.845	0.618	0.659
40	**0.552**	**0.804**	**0.898**	**0.641**	**0.682**
4$\hat{0}$	0.542	0.775	0.862	0.622	0.666
50	**0.538**	**0.794**	**0.906**	**0.632**	**0.673**
5$\hat{0}$	0.525	0.775	0.881	0.617	0.661
100	**0.528**	**0.809**	**0.935**	**0.629**	**0.667**
1$\hat{0}$0	0.511	0.833	0.918	0.614	0.656
1000	**0.462**	**0.814**	**0.973**	**0.571**	**0.618**
10$\hat{0}$0	0.453	0.801	0.954	0.567	0.611

Table 5: The influence of using multiple prompt templates on the experiment

	S@1	**S@3**	**S@10**	**MAP**	**MRR**
30_{method}	0.552	0.782	0.847	0.623	0.665
50_{method}	**0.542**	0.789	0.884	0.631	0.670
45_{class}	0.678	**0.903**	**0.959**	0.758	0.787
50_{class}	0.682	**0.906**	**0.969**	**0.763**	0.791

the prompt template in front is always the best. Table 5 is to verify whether the more templates we add, the better the experimental results will be. In this experiment, we find 10 similar problems as the initial template in the first step. Then, in the range of 10, we set three different random numbers as different three prompt templates for experiments. We found that there are more experimental results than a single template at both the method level and the class level.

After the analysis of the above results, first of all, for the study of the position of the prompt template, we believe that the use of the word bag model without order, the order before and after will not affect the experimental results. After subsequent research, it is found that when the user 's question is combined with the prompt template, the statement will be too long. In the experiment, we set the maximum statement length, and some newly generated statements will lose key information due to too long, thus affecting the accuracy. The prompt template is more valuable for the user 's display intention. As for the class level, we find that the problems on the class level are relatively short, so the two values are consistent. For multi-templates, our initial guess is that the more prompts, the better the experimental results, because the use of the bag-of-words model, the more prompts can provide more

keywords for query, so the results will be better. But in the end, it was found that the experimental results did not have obvious advantages. Later, we found that the experimental results of multiple templates have a particularly high quality requirement for user input problems. As shown in Figure 1 (b), we enter the problem of ' Using Java to Sort Content '. Due to the particularly poor quality of the problem, multiple templates will give keywords with very different meanings, resulting in a particularly poor recommendation effect.

The experimental results show that if the sentence length based on the bag-of-words model is set very large, the position of the prompt template has no effect on the experiment. The influence of multi prompt templates on the experiment is affected by the quality of user input problems.

6 DISCUSSION

In this section, we will discuss the implications of using prompt learning to recommend APIs to explore other possible directions, as well as potential threats in experimental approaches.

6.1 Implications

Using prompt Learning to enhance API recommendations : Researchers have demonstrated the effectiveness of using a bag-of-words-based word embedding model for API recommendation. However, there are usually two problems in the bag-of-words model. (1) The approach uses bag-of-words and embeds words, but it cannot capture the order information. (2) The two words in the bag of words are similar in grammar, but there may be a big difference in grammar, so only using the context of the word is not enough to distinguish between queries. We use prompt learning to visualize the user 's needs, and can enhance the semantics of the query keywords when performing similarity calculations, which can reduce the shortcomings of using the bag-of-words model for recommendation. Therefore, we suggest that researchers can combine sequential and semantic information to further study the effectiveness of prompt learning.

Selection of approaches in prompt learning : With the development of the times, prompt learning has become a new favorite in the direction of NLP. The prompt is a paradigm or template designed by the researcher for the downstream task, which makes the downstream task to accommodate the pre-training model. The purpose is to make the downstream task similar to the pre-training, so that the potential of the pre-training model can be brought into play. However, there are usually two problems when using prompt learning : (1) How to select a template ? (2) The token predicted by the model is sometimes uncontrollable and difficult to map to the label. At present, there are two template selection methods, in which one is called Hard-prompt artificial construction template, the other is Soft-prompt automatic learning template. In this paper, we use artificially constructed templates to artificially control the location of

the template. It is hoped that the following researchers can use Soft-prompt in API recommendation research.

6.2 Threats to Validity

Internal Validity. In this paper, we mainly study the improvement effect of API recommendation combined with prompt learning. In terms of datasets, we use the dataset used by the BIKER author 's team, and do not collect relevant data by ourselves. However, we view this data set to determine the high quality of the data set and can be used on our model. Therefore, this problem can be ignored.

External Validity. On the one hand, we use prompt learning in the API recommendation model. At present, there is no particularly good solution to the two problems in prompt learning. On the other hand, we use the bag-of-words model in the API recommendation stage, and the bag-of-words model also has some defects. However, the purpose of this paper is to study the simultaneous prompts to show the user 's willingness to improve the recommendation effect, so these shortcomings can be ignored. At the same time, our method model only takes into account the JAVA language, and lacks research on other languages.

7 RELATED WORK

In this section, we will review the existing work on prompt learning in the API recommendation direction. The existing work on prompt learning in API recommendation direction can be divided into two categories : API recommendation research direction and prompt learning research direction.

API recommendation : At present, many researchers have studied API recommendation, which is mainly divided into recommendation based on existing code context information and recommendation based on user input. For the first direction of research, Zhong et al. proposed that MAPO [8] mines and clusters API usage patterns from open source databases, and then recommends relevant usage patterns to users. Nguyen et al.proposed APIREC [24], which recommends APIs by changing the corresponding context and fine-grained code. Fowkes et al. proposed PAM [10], which solves the problem of large API recommendation list by mining API usage patterns through a probabilistic algorithm with few parameters. Nguyen et al. proposed FOCUS [11], which finds API usage in similar projects from an open source knowledge base, and implements API recommendation through context-based collaborative filtering [25, 26] technology. Xie et al. proposed GAPI [12], which proposes a new graph-based API recommendation method. This method uses GNN [27] to capture collaborative signals from API call interactions and project structures, and these interactions constitute graphs for API usage recommendation. Chen et al. proposed MEGA [13], which uses a structure-aware attention network and a frequency-aware attention network to construct a multi-view heterogeneous graph representation model between the project and the API method, thereby increasing API interaction.

For the second direction, Zhang et al. proposed PASH [28]. This method obtains the API sequence according to the literature, and then uses the information in SO to reorder the sequence, and finally obtains the API result. Rahman et al. proposed

RACK [14], which recommends API lists for natural language queries by constructing keyword-API association information in SO. Huang et al. proposed BIKER [15], they believed that there is a knowledge gap between the user 's input and the knowledge base, so they proposed the concept of double information sources, and sorted the API through multiple information sources to achieve API recommendation at the method level and class level. Zhou et al. proposed the framework of BRAID [16], which uses user feedback to continuously improve the performance of API recommendation. In the field of deep learning, Gu et al. proposed DeepAPI [29], which uses neural networks to transform API recommendation tasks into encoding and decoding tasks. Wei et al. proposed CLEAR [30], which uses BERT [31] sentence embedding and comparative learning methods to solve the word order and semantic problems when user input problems, thereby improving the API recommendation effect. Irsan et al. proposed PICASO [17] on the basis of CLEAR, which proposes to convert the problem into multiple input sources to improve the recommendation effect before inputting the problem into the model. Compared with BIKER and BRAID, PTAPI can better express the real intention of users, so it is easier to get close to the answer in the knowledge base.

Prompt learning: Recently, the breakthrough of self-supervised [32] pre-trained language model has prompted the development of natural language processing. Radford et al. proposed GPT [33]. This method first uses the Transformer architecture to pre-train large-scale network text. Since Brown et al.proposed GPT-3 [34], prompt learning has received great attention. This method enables large-scale language models to achieve excellent performance in low data states through context learning and instant tuning. But it relies heavily on manual prompts to serve downstream tasks. Shin et al. [35] used a token-based gradient search, and Gao et al. [36] used a separate model to automatically search for discrete prompts. Li & Liang [37] proposed a prefix tuning method for natural language generation tasks, which can train continuous prompts. PTAPI relies on manual selection of templates and manual tuning because our model does not require too many parameters.

8 CONCLUSION

In this paper, we introduce PTAPI, a novel API recommendation method that leverages information from Stack Overflow to visualize the user's genuine intention and facilitate the selection of an appropriate prompt template. By combining the prompt template with the user's problem description, a new input problem is generated. This newly generated problem serves the dual purpose of clarifying the user's true intention and enhancing the expressiveness of the statement. Through our experimental evaluation, we demonstrate the effectiveness of PTAPI at both the method and class levels.

In the future, we will expand the scope of PTAPI, on the one hand, it can be applied to more programming languages. On the other hand, we will choose more prompt learning to apply to more model methods. Finally, we will make unremitting efforts to improve API recommendation performance.

References

[1] Thung, F.: Api recommendation system for software development. In: Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering, pp. 896–899 (2016)

[2] Hou, D., Yao, X.: Exploring the intent behind api evolution: A case study. In: 2011 18th Working Conference on Reverse Engineering, pp. 131–140 (2011). IEEE

[3] Yu, Z., Bai, C., Seinturier, L., Monperrus, M.: Characterizing the usage, evolution and impact of java annotations in practice. IEEE Transactions on Software Engineering **47**(5), 969–986 (2021)

[4] Gvero, I.: Core java volume I: fundamentals, 9th edition by cay s. horstmann and gary cornell. ACM SIGSOFT Softw. Eng. Notes **38**(3), 33 (2013)

[5] Oracle: Jdk 18 documentation, (2022). https://docs.oracle.com/en/java/javase/18/books.html

[6] Sacramento, P., Cabral, B., Marques, P.: Unchecked exceptions: Can the programmer be trusted to document exceptions. IVNET'06 (2006)

[7] Li, Z., Wu, J., Li, M.: Study on key issues in api usage. Journal of software **29**(6), 1716–1738 (2018)

[8] Xie, T., Pei, J.: Mapo: Mining api usages from open source repositories. In: Proceedings of the 2006 International Workshop on Mining Software Repositories, pp. 54–57 (2006)

[9] Wang, J., Dang, Y., Zhang, H., Chen, K., Xie, T., Zhang, D.: Mining succinct and high-coverage api usage patterns from source code. In: 2013 10th Working Conference on Mining Software Repositories (MSR), pp. 319–328 (2013). IEEE

[10] Fowkes, J., Sutton, C.: Parameter-free probabilistic api mining across github. In: Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering, pp. 254–265 (2016)

[11] Nguyen, P.T., Di Rocco, J., Di Ruscio, D., Ochoa, L., Degueule, T., Di Penta, M.: Focus: A recommender system for mining api function calls and usage patterns. In: 2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE), pp. 1050–1060 (2019)

[12] Ling, C., Zou, Y., Xie, B.: Graph neural network based collaborative filtering for api usage recommendation. In: 2021 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER), pp. 36–47 (2021). IEEE

[13] Chen, Y., Gao, C., Ren, X., Peng, Y., Xia, X., Lyu, M.R.: API usage recommendation via multi-view heterogeneous graph representation learning. IEEE Trans.

Software Eng. **49**(5), 3289–3304 (2023)

[14] Rahman, M.M., Roy, C.K., Lo, D.: RACK: automatic API recommendation using crowdsourced knowledge. CoRR **abs/1807.02953** (2018)

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

[16] Zhou, Y., Yang, X., Chen, T., Huang, Z., Ma, X., Gall, H.: Boosting api recommendation with implicit feedback. IEEE Transactions on Software Engineering **48**(6), 2157–2172 (2022)

[17] Irsan, I.C., Zhang, T., Thung, F., Kim, K., Lo, D.: Picaso: Enhancing api recommendations with relevant stack overflow posts. 2023 IEEE/ACM 20th International Conference on Mining Software Repositories (MSR), 92–37 (2023)

[18] Mikolov, T., Sutskever, I., Chen, K., Corrado, G.S., Dean, J.: Distributed representations of words and phrases and their compositionality. Advances in neural information processing systems **26** (2013)

[19] Loper, E., Bird, S.: Nltk: The natural language toolkit. arXiv preprint cs/0205028 (2002)

[20] Lam, A.N., Nguyen, A.T., Nguyen, H.A., Nguyen, T.N.: Combining deep learning with information retrieval to localize buggy files for bug reports (n). In: 2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE), pp. 476–481 (2015)

[21] Zanjani, M.B., Kagdi, H., Bird, C.: Automatically recommending peer reviewers in modern code review. IEEE Transactions on Software Engineering **42**(6), 530–543 (2015)

[22] Wen, M., Wu, R., Cheung, S.-C.: Locus: Locating bugs from software changes. In: Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering, pp. 262–273 (2016)

[23] Saha, R.K., Lease, M., Khurshid, S., Perry, D.E.: Improving bug localization using structured information retrieval. In: 2013 28th IEEE/ACM International Conference on Automated Software Engineering (ASE), pp. 345–355 (2013)

[24] Nguyen, A.T., Hilton, M., Codoban, M., Nguyen, H.A., Mast, L., Rademacher, E., Nguyen, T.N., Dig, D.: Api code recommendation using statistical learning from fine-grained changes. In: Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering, pp. 511–522 (2016)

[25] Sarwar, B., Karypis, G., Konstan, J., Riedl, J.: Item-based collaborative filtering recommendation algorithms. In: Proceedings of the 10th International Conference on World Wide Web, pp. 285–295 (2001)

[26] Su, X., Khoshgoftaar, T.M.: A survey of collaborative filtering techniques. Advances in artificial intelligence **2009** (2009)

[27] Scarselli, F., Gori, M., Tsoi, A.C., Hagenbuchner, M., Monfardini, G.: The graph neural network model. IEEE transactions on neural networks **20**(1), 61–80 (2008)

[28] Zhang, J., Jiang, H., Ren, Z., Chen, X.: Recommending apis for api related questions in stack overflow. IEEE Access **6**, 6205–6219 (2017)

[29] Gu, X., Zhang, H., Zhang, D., Kim, S.: Deep api learning. In: Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering, pp. 631–642 (2016)

[30] Wei, M., Harzevili, N.S., Huang, Y., Wang, J., Wang, S.: Clear: contrastive learning for api recommendation. In: Proceedings of the 44th International Conference on Software Engineering, pp. 376–387 (2022)

[31] Devlin, J., Chang, M.-W., Lee, K., Toutanova, K.: Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805 (2018)

[32] Liu, X., Zhang, F., Hou, Z., Mian, L., Wang, Z., Zhang, J., Tang, J.: Self-supervised learning: Generative or contrastive. IEEE transactions on knowledge and data engineering **35**(1), 857–876 (2021)

[33] Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I., *et al.*: Language models are unsupervised multitask learners. OpenAI blog **1**(8), 9 (2019)

[34] Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J.D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., *et al.*: Language models are few-shot learners. Advances in neural information processing systems **33**, 1877–1901 (2020)

[35] Shin, T., Razeghi, Y., Logan IV, R.L., Wallace, E., Singh, S.: Autoprompt: Eliciting knowledge from language models with automatically generated prompts. arXiv preprint arXiv:2010.15980 (2020)

[36] Gao, T., Fisch, A., Chen, D.: Making pre-trained language models better few-shot learners. arXiv preprint arXiv:2012.15723 (2020)

[37] Li, X.L., Liang, P.: Prefix-tuning: Optimizing continuous prompts for generation. arXiv preprint arXiv:2101.00190 (2021)