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. To deal with this uncertainty, we adopt the method of prompt learning to help users express their real needs more clearly.

Prompt learning [18, 19] is a new paradigm in natural language processing. It can design a series of prompts to guide the model to better understand and process tasks. Among them, the template is a framework for building prompts. Through the design of the template, the potential of the upstream pre-training model can be tapped.

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 enters the problem to be solved, finds similar problems from the SO post, and the developer selects the most similar problem as the prompt template, and combines the two sentences into a new input through a connection process.

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.

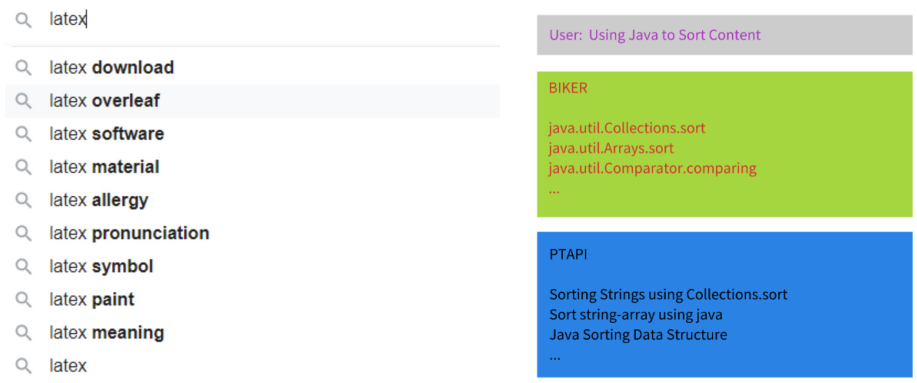
The rest of this paper is organized as follows. The Section 2 describes the motivation example of this paper. The Section 3 elaborates our approach. The Section 4 introduces the evaluation settings, and the Section 5 introduces the experimental results. The Section 6 shows our user study. The Section 7 discusses our work, and the Section 8 introduces the related work. Finally, the Section 9 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[20, 21]. 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.



(a) Example of recommendation sequences from browser

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

Fig. 1 Example diagram

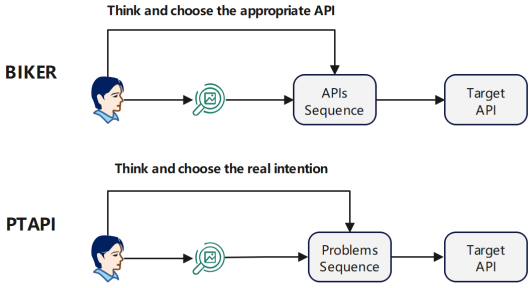


Fig. 2 Recommendation idea diagram

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, 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 subsection, first, we need to use SO posts to build a corpus for calculating the similarity between queries and SO posts or API descriptions. Get the text content from the SO posts in the HTML page and process it. Because some SO posts contain long code fragments, this will increase the burden of training. Therefore, we delete the posts

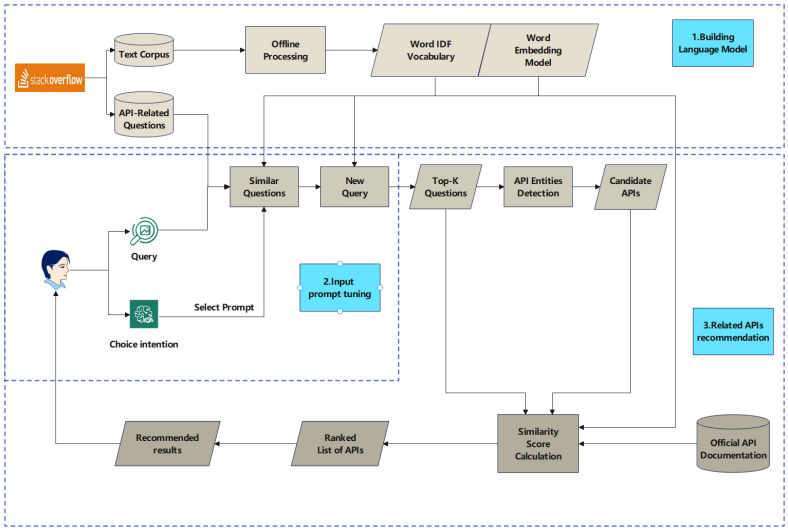


Fig. 3 Overall framework of PTAPI

containing long code fragments, retain the posts containing short code fragments, and use the NLTK package [22] to label the sentences. The NLTK package can identify the part of speech of each word in the sentence. Secondly, we need to use Word2Vec [23] to train a word embedding model, which is the basic model for measuring word similarity. Then, we construct the word IDF (inverse document frequency) vocabulary based on the SO corpus. The IDF of the word represents the reciprocal of the number of posts containing the word. Among them, the more posts a word appears in, the less likely it is to carry important semantic information, so its IDF value is lower. Each word in the text is abbreviated into a root form. For instance, the stem of 'running', 'runs', and 'run' is 'run', and the stem of 'am', 'is', and 'are' is 'be' Words with the same root have the same IDF value. Finally, because the amount of text in the SO corpus is much larger than that in the API document, the words in the API document directly select the word embedding model and the IDF vocabulary.

3.3 Generate prompt template

In this subsection, PTAPI generates prompts based on the user 's input, and the newly generated prompts can better express the user 's true intention. First, through the similarity language model constructed in the previous section, PTAPI finds several titles that are most similar to the user input from the SO problem library. We use formula 1 to calculate the similarity between the query and the SO problem library :

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

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 subsection, we find the prompt template according to the user input description, and combine the generated template with the user input description into a new input statement. Continue the operation of the previous stage to calculate the similarity between the newly generated sentence and the sentence in the SO problem library. Through calculation, we find k problems that are most similar to the input from the SO problem library. After retrieving the top-k similar questions, PTAPI uses two heuristic rules to retrieve APIs from each question. One detects the full name of the corresponding API method from the hyperlink using a regularized expression by using each hyperlink in each answer. Another uses a dictionary to store the names of all APIs from official documents[24]. Then, PTAPI checks the plain text contained in the HTML tags in each answer. If the API in the plain text matches the one in the dictionary, it is marked as a candidate API.

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 $Q'_q = Q_q \cup max(sim(Q_{so}, Q_q))$ represents the newly generated problem description, The symbol $\cup$ indicates that the user 's original input is connected to the newly generated prompt template. 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 test and evaluate the effectiveness of PTAPI, we reuse BIKER. BIKER downloaded the official data dump [25] of SO (published in: Dec 9th, 2017) as its own SO text corpus. Next, we need an SO problem library to implement the second stage of prompt template generation and the third stage of similar problem retrieval. Because SO is an open question and answer website, the quality of the answers is often not high. Therefore, these data need to be strictly screened. It has the following screening criteria : 1) All scores in the library should be positive. 2) Each question has at least one API answer, and the score of the answer also needs to be positive. Finally, we need to select a test dataset to evaluate the effectiveness of PTAPI. This requires higher quality of data, and we need to ensure that the API in the answer is the correct API. The screening criteria for the test database are as follows : 1) The score of the problem itself needs to be greater than 5 points. 2) There must be an answer in the question, and the score is positive. 3) The data is not in the SO problem library. In this way, a preliminary dataset is obtained. After manual labeling, a test dataset containing 413 problems and their ground truth API is obtained.

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 [26–29]. 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. When searching for top-k similar problems to detect candidate APIs, we note that the number of retained similar problems has an important impact on the recommendation results. If you retain too few similar problems, you may miss the target API required by the user, and if you retain too much, you may introduce too much noise data. In subsection 5.1, we set the parameter to 50 (retaining 50 similar questions) to compare with BIKER and BRAID. In subsection 5.2, we manually adjust the size of the parameters to verify the impact of the number of similar problems on the experiment. In subsection 5.3, according to the user 's input, we feedback to the user 10 prompts (10 prompts can be found in the probability of meeting the user 's intentions) as a user selection prompt template.

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

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**

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.

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 in S @ 1 at the method level and 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 prompts can help users better choose APIs. We find that BRAID accepts the user 's initial input and the results recommended by the existing API as input on the basis of BIKER. It uses user history selection as feedback information and uses active learning technology to establish a new API recommendation model. This makes BRAID may be overly dependent on historical data and vulnerable to initial data bias. Meanwhile, the choice of user history may not fully reflect their long-term intentions. PTAPI can give users prompts in real time, and these prompts can also best reflect the user 's most real intentions at present, so our method can be greatly improved in accuracy.

PTAPI has a great advantage compared with other baselines, which indicates that the display information has a positive effect on improving the accuracy of API recommendation.

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.

The parameter here refers to the number of similar SO posts retained in the recommendation results, and the candidate API is obtained by retrieving the answers to the questions 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.

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

By analyzing the data in the table, we can find that the performance of PTAPI will first improve and then decrease with the increase of parameters. This is because in the beginning, we kept fewer similar questions, and the scope of the search was not large enough, which caused a lot of important relevant API information to be lost. With the increase of similar problems, the accuracy of the model will gradually improve. However, when the performance of the model reaches the optimal value, the introduction of similar problems will lead to the increase of a large number of noisy data, and the performance of the model will gradually decrease.

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 The influence of the position of the template on the experiment was suggested

	S@1	**S@3**	**S@10**	**MAP**	**MRR**
30	**0.561**	**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

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 the prompt template in front is always the best. Table 5 is to verify whether the more templates are added, the better the experimental results. In this experiment, we first get 10 prompts based on the user ' s input. Then, three different prompts were randomly selected within the range of 10 prompts to combine with the original input

and form a new input. The experiment was repeated 10 times in each group, and the final results were averaged.

After analyzing the above results, first of all, for the study of the position of the prompt template, we initially thought that the order of words in the bag-of-words model would not affect the experimental results. After subsequent research, we found that we use the continuous word bag model, and set the size of the window to 5, the original input is connected to the prompt word, and the two sentences will affect each other. The experimental results show that it is better to put the prompt in front of the text. Secondly, for using multiple prompt templates, our initial guess is that the more tips, the better the experimental results. Because we use the bag of words model, the more tips can provide more keywords to query, which can enhance the expression of words, so as to achieve better results. Meanwhile, multiple prompts may provide multiple different keywords, and multiple keywords can be combined into the user 's true intention. 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 encountered the problem of ' using Java to sort content '. Due to the poor quality of the problem, multiple templates will give keywords with very different meanings, resulting in a poor recommendation effect.

The experimental results show that, first, putting the prompt in front of the sentence position will get better results. Second, the number of input prompts is not the more the better.

6 USER STUDY

In this section, we will conduct user research to verify whether it can help users find appropriate APIs more effectively and accurately by allowing users to use PTAPI.

6.1 Study Design

Experimental Queries and Ground-Truth APIs. We randomly selected 10 questions from the test dataset, as shown in Table 6, where the last column shows the ground-true answer to the question.

Participants. We recruited 12 participants (6 college students and 6 graduate students) from the university where the first author worked. These college students have 2 to 3 years of JAVA learning experience, and these graduate students have JAVA development experience in graduation design and some internship projects.

Experimental Groups. Next, we divide these participants into three groups on average, set as follows : 1) SOW: Find appropriate API methods by **S**earching and querying resources **O**n the **W**eb ; 2) BIKER : Use BIKER to search and query answers. 3) PTAPI : Use PTAPI to search and query for answers.

Table 6: Experimental questions and corresponding answers for user study

ID	Query	Answer
Q1	Make a negative number positive?	java.lang.Math.abs
Q2	How can I stop a Java while loop from eating >50% of my CPU?	java.lang.Thread.sleep
Q3	How to convert date format to milliseconds?	java.util.Date.getTime
Q4	How can I write to a specific line number in a txt file in Java?	ava.nio.file.Files.write
Q5	Programmatically determine which Java thread holds a lock?	java.lang.Thread.holdsLock
Q6	Handling passwords used for auth in source code?	javax.swing.JPasswordField.getText
Q7	How to blur a portion of an image with JAVA?	java.awt.image.BufferedImage.getSubimage
Q8	How to get string value from a Java field via reflection?	java.lang.reflect.Field.get
Q9	Best way to exit a program when I want an exception to be thrown?	java.lang.System.exit
Q10	How to get concrete type of a generic interface?	java.lang.Class.getGenericInterfaces

Table 7:Results of user study

Evaluation Metrics	Different groupings	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	Average
Correctness	SOW	**1**	0.75	0.75	0.5	0.75	0.5	0.25	0.5	0.25	0.25	0.55
	BIKER	**1**	**1**	**1**	0.5	**1**	0.5	**0.5**	0.5	0.75	**0.5**	0.725
	PTAPI	**1**	**1**	**1**	**0.75**	**1**	**0.75**	**0.5**	**1**	**1**	**0.5**	**0.85**
Completion time	SOW	53s	185s	112s	214s	174	232s	248s	182s	289s	165s	185.4s
	BIKER	**40s**	**102s**	**95s**	156s	**120s**	196s	199s	140	230s	152s	143s
	PTAPI	46s	124s	105s	**145s**	156s	**183s**	**154s**	**124s**	**219s**	**140s**	**139.6s**

6.2 Results Analysis

We analyze two metrics based on the user 's experimental results, as follows :

·**Correctness** : This indicator evaluates whether participants can find the correct API. If the correct API is obtained, the correctness is 1, otherwise it is 0. We found that there may be more than one answer to some questions, such as java.lang.System.currentTimeMillis can also solve the Q2 problem. Therefore, we need to manually check the answers of each participant and set the correctness of these correct answers to 1.

·**Completiontime** : This indicator is used to evaluate the speed at which participants respond to questions after using the tool. In this experiment, we selected students with poor programming ability to conduct experiments. By asking participants, we find that it is difficult for them to directly give the correct answer to the above questions. Therefore, for each problem, our final time spent selects the average of each group.

Table 7 shows the user 's study. From the table, we can see that our approach is generally better than the other two approaches in correctness. In terms of completion time, we found that direct query on the network is the slowest approach. Among these 10 questions, BIKER has 4 questions that take less time than us. Combining the correctness aspect, we find an interesting phenomenon : BIKER takes less time when the query is relatively simple. By asking participants, we know that participants using PTAPI need to spend some time in selecting the prompt template.

In short, PTAPI is effective for users, and the more difficult the problem is for users, the more it can reflect the performance of our approach.

7 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.

7.1 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 bag-of-words model only focuses on the presence or absence of words, and cannot capture the contextual relationship between words. Context is important for understanding the meaning of text. 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.

7.2 Threats to Validity

Internal Validity. In this paper, we generate prompts and form new problems based on user input, and combine SO posts and API documents for similarity calculation to obtain the final API ranking. Due to the variety of APIs and the rapid update of their versions, PTAPI is not very comprehensive in considering third-party libraries and API quality issues. Meanwhile, the model using a regularized expression to extract hyperlinks and API names may be error-prone. However, this paper mainly studies the impact of prompt learning on API recommendation. The training dataset and the test dataset we used are of high quality and the collected time is similar, so these threats do not affect our experiment.

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.

Meanwhile, our method model only takes into account the JAVA language, and lacks research on other languages.

8 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 [30], 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 [31, 32] technology. Xie et al. proposed GAPI [12], which proposes a new graph-based API recommendation method. This method uses GNN [33] 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 [34]. 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 [35], which uses neural networks to transform API recommendation tasks into encoding and decoding tasks. Wei et al. proposed CLEAR [36], which uses BERT [37] 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 [38] pre-trained language model has prompted the development of natural language processing. Radford et al. proposed GPT [39]. This method first uses the Transformer architecture to pre-train large-scale network text. Since Brown et al.proposed GPT-3 [40], 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. [41] used a token-based gradient search, and Gao et al. [42] used a separate model to automatically search for discrete prompts. Li & Liang [43] 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.

9 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] Liu, P., Yuan, W., Fu, J., Jiang, Z., Hayashi, H., Neubig, G.: Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. ACM Comput. Surv. **55**(9), 195–119535 (2023)

[19] Wang, X., Zhou, K., Wen, J.-R., Zhao, W.X.: Towards unified conversational recommender systems via knowledge-enhanced prompt learning. In: Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, pp. 1929–1937 (2022)

[20] Ponzanelli, L., Mocci, A., Bacchelli, A., Lanza, M., Fullerton, D.: Improving low quality stack overflow post detection. In: 2014 IEEE International Conference on Software Maintenance and Evolution, pp. 541–544 (2014)

[21] Liu, K., Chen, X., Chen, C., Xie, X., Cui, Z.: Automated question title reformulation by mining modification logs from stack overflow. IEEE Trans. Software Eng. **49**(9), 4390–4410 (2023)

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

[23] 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)

[24] Java se 8 api documentation downloading site. (2017). http://www.oracle.com/technetwork/java/javase/documentation/jdk8-doc-downloads-2133158.html

[25] Stack overflow data dump. (2017). https://archive.org/download/stackexchange

[26] 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)

[27] 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)

[28] 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)

[29] 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)

[30] 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)

[31] 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)

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

[33] 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)

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

[35] 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)

[36] 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)

[37] 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)

[38] 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)

[39] 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)

[40] 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)

[41] 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)

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

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