Deep API Sequence Generation via Golden Solution Samples and API Seeds

YUEKAI HUANG*, Institute of Software Chinese Academy of Sciences, China
JUNJIE WANG*†, Institute of Software Chinese Academy of Sciences, China
SONG WANG, York University, Canada
MOSHI WEI, York University, Canada
LIN SHI, Beihang University, China
ZHE LIU*, Institute of Software Chinese Academy of Sciences, China
QING WANG*†, Institute of Software Chinese Academy of Sciences, China

Automatic API recommendation can accelerate developers' programming, and has been studied for years. There are two orthogonal lines of approaches for this task, i.e., information retrieval-based (IR-based) approaches and sequence to sequence (seq2seq) model based approaches. Although these approaches were reported to have remarkable performance, our observation finds two major drawbacks, i.e., IR-based approaches lack the consideration of relations among the recommended APIs, and seq2seq models do not model the API's semantic meaning. To alleviate the above two problems, we propose APIGens, which is a retrieval-enhanced large language model (LLM) based API recommendation approach to recommend an API sequence for a natural language query. The approach first retrieves similar programming questions in history based on the input natural language query, and then scores the results based on API documents via a scorer model. Finally, these results are used as samples for few-shot learning of LLM. To reduce the risk of encountering local optima, we also extract API seeds from the retrieved results to increase the search scope during the LLM generation process. The results show that our approach can achieve 48.41% ROUGE@10 on API sequence recommendation and the 82.61% MAP on API set recommendation, largely outperforming the state-of-the-art baselines.

CCS Concepts: • **Software and its engineering → Software notations and tools**.

Additional Key Words and Phrases: API recommendation, deep learning, information retrieval, sequence generation, large language model

*Also With State Key Laboratory of Intelligent Game, Beijing, China;
Science and Technology on Integrated Information System Laboratory, Beijing, China;
University of Chinese Academy of Sciences, Beijing, China;

†Corresponding author

Authors' addresses: Yuekai Huang, huangyuekai18@mails.ucas.ac.cn, Institute of Software Chinese Academy of Sciences, Beijing, China; Junjie Wang, junjie@iscas.ac.cn, Institute of Software Chinese Academy of Sciences, Beijing, China; Song Wang, wangsong@yorku.ca, York University, Toronto, ON, Canada; Moshi Wei, moshiwei@yorku.ca, York University, Toronto, ON, Canada; Lin Shi, shilin@buaa.edu.cn, Beihang University, Beijing, China; Zhe Liu, liuzhe181@mails.ucas.ac.cn, Institute of Software Chinese Academy of Sciences, Beijing, China; Qing Wang, wq@iscas.ac.cn, Institute of Software Chinese Academy of Sciences, Beijing, China.

ACM 1557-7392/2024/9-ART
https://doi.org/10.1145/3695995

Table 1. Motivation examples of API recommendation

Query: How to acess list of files and folders in a zip file in java?		
Accepted Answer	**BIKER results**	**DeepAPI results**
java.util.zip.ZipFile.entries	java.util.zip.ZipFile.entries ●	Sequence 1:
java.util.Enumeration.hasMoreElements	java.util.jar.JarFile.entries ●	(java.io.File.listFiles
java.util.Enumeration.nextElement	java.nio.file.Files.copy	java.nio.file.Paths.get)
java.util.zip.ZipEntry.getName	java.io.File.listFiles ◇	Sequence 2:
	java.nio.file.Files.walk ○	(java.io.File.listFiles
	java.io.File.listRoots ◇	java.io.File.isDirectory
	java.util.zip.ZipFile.getInputStream	java.io.File.listFiles)
	java.nio.file.Files.walkFileTree ○	Sequence 3:
	java.nio.file.Files.newDirectoryStream	(java.io.File.listFiles
	java.nio.file.Files.createDirectories	...

1 INTRODUCTION

Over the past decades, open-source software development has received extensive attention from the software engineering community. This attention leads to a tremendous demand for already devised libraries or APIs (Application Programming Interfaces) which facilitate software development and maintenance. While there are many publicly available open-source resources, searching the right APIs for specific tasks is not easy, especially for the plenty of green hands [45]. This can be partially demonstrated by the dozens of new emerging questions per day in the Question&Answer website as Stack Overflow (SO).

To help with API search, many automated API recommendation approaches have been proposed [9, 14, 40, 42], which recommend a set/sequence of APIs for a natural language query. There are two orthogonal lines of approaches for this task, i.e., information retrieval (IR) based approaches [14, 40, 42] and sequence to sequence (seq2seq) model based approach[9]. Typical IR-based approach, for example BIKER [14], mainly utilizes word embedding related techniques to calculate the similarity between the query and existing SO posts and recommend a set of APIs from the answers of the most similar SO posts. The representative seq2seq approach is DeepAPI [9], and it utilizes the Recurrent Neural Network (RNN) network to translate from the natural language query to API sequences. Although these approaches achieved remarkable performance on their own datasets, we find two major problems that can affect their effectiveness.

The first problem is that the IR-based approaches do not consider the relevance between multiple APIs and query, resulting in recommending similar APIs and subsequently missing other needed APIs. As shown in Table 1, for the query *"How to acess list of files and folders in a zip file in java?"*, the recommended APIs by BIKER contain several similar APIs (i.e., marked ones), e.g., *java.util.zip.ZipFile.entries* and *java.util.zip.JarFile.entries*. This is mainly because the IR-based approaches try to find the correct APIs by matching the query and the API documentation, and the APIs similar to the ground-truth APIs would exert higher similarity with the query, and therefore tend to be recommended. However, the higher ranking of these similar APIs would subsequently lower the ranking of (or miss) other needed APIs, e.g., *java.util.Enumeration.hasMoreElements* in Table 1. Although some IR based approaches [40, 42] can recommend SO posts or API sequences, its essence is to model query and programming questions. Most importantly, its recommendation scope is limited by the database and cannot cover all possible situations, especially for rare API or its combination, as these APIs are likely not to appear in historical programming questions.

On the contrary, the seq2seq-based approaches can consider multiple APIs as a whole to generate API sequences. Nevertheless, the problem with these approaches is that they only utilize a query's context information for generating the API sequences, yet do not consider the semantic meaning of the recommended APIs, resulting

in incorrect API sequence. As shown in Table 1, the APIs generated by DeepAPI include incorrect APIs like *java.io.File.listFiles* and *java.nio.file.Paths.get*, while both of them are widely used for IO read processing, rather than zip file operation as queried in the example. This is mainly because it does not fully understand the semantic meaning of the APIs and the query.

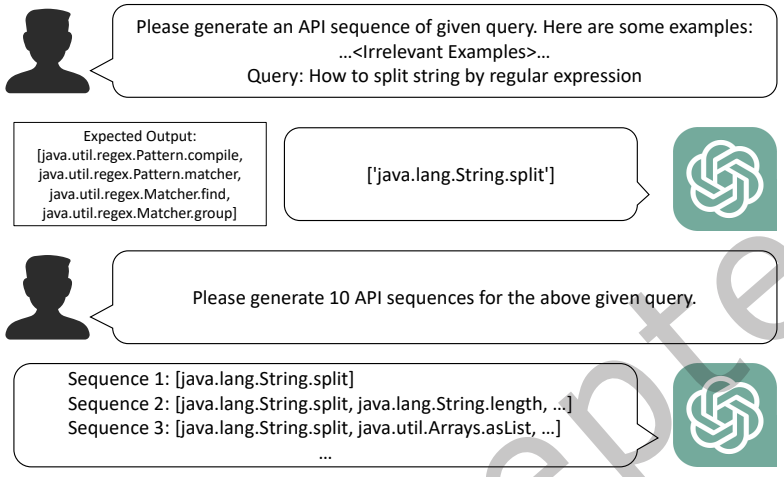


Fig. 1. Fail example of API sequences generated by LLM

In addition to the previously mentioned approaches, LLM can also be used for API recommendation tasks. However, relying solely on LLM may face two challenges. As shown in Figure 1, **the first challenge is how to find suitable examples.** For LLM, using examples as prompts can often improve the performance, but inappropriate examples may mislead the model. Therefore, finding suitable examples is a challenge that needs to be addressed. **The second challenge is how to recommend more reasonable results**, which mainly suffer from the generation strategy of LLM. For a recommendation task, due to the inability to guarantee the correctness of the model, multiple results are often recommended for users to choose from. In order to make LLM generate multiple results, we usually use the parameter *n* or actively mention the requirements in the prompt. However, LLM commonly use greedy strategies for generation, although it is possible to increase the randomness of a LLM by adjusting parameters such as *temperature*, the generated tokens are also sampled based on the greedy strategy, which can easily result in encountering local optima. In addition, the LLM may have unstable output, which may lead to the LLM selecting recommendation results from the vicinity of local optima which makes the output results very similar, and when the initial result is incorrect, subsequent results are difficult to rectify, resulting in incorrect outputs. Therefore, using LLM alone is not sound, which is why we need to design APIGens to complete this task.

To alleviate the above problems, we propose APIGens, which is a retrieval-enhanced LLM-based API recommendation approach to recommend API sequences for a natural language query. In this paper, we define the problem as generating multiple API sequences for a given natural language query. We mainly consider generative solutions because this task is different from single API recommendations, as the number of API sequences far exceeds the number of APIs. This approach is motivated by a common development phenomenon, that is, when the developers encounters a programming question, they will search whether there is a matching solution through the Q&A website like Stack Overflow, and when they can not find a suitable solution, they will use the existing

similar solutions as a reference. For the unfamiliar API in the solutions, they will query the specific usage through the API document. Finally, based on the obtained information, write a solution that meets their requirements.

In our proposed approach, we combine the advantage of both IR-based approach and generative approach. In detail, in order to fully utilize historical data and API documents, we used IR-based approach to retrieve similar solutions. Meanwhile, to address the APIs recommendation drawback of IR-based approach, we introduced LLM for API sequence generation. This IR&LLM based approach can also address the challenge of LLM in finding suitable examples. For the second challenge of LLM, we designed API seeds extraction, which is to extract the most relevant APIs from the examples and ask LLM to generate an API sequence for each API seed. This can appropriately increase the generation scope of LLM without introducing noise as much as possible, thereby reducing the risk of LLM encountering local optima.

The evaluation is conducted on the commonly-used dataset for API set recommendation and a newly collected dataset for API sequence recommendation. Results demonstrate that, for API sequence recommendation, APIGens can achieve ROUGE@10 of 48.41%, outperforming the state-of-the-art approaches by 57.68% to 125.30% on method level. For API set recommendation, APIGens can achieve Recall@10 of 96.33% and MAP of 82.61%, outperforming the state-of-the-art approaches by 5.50% to 53.36%.

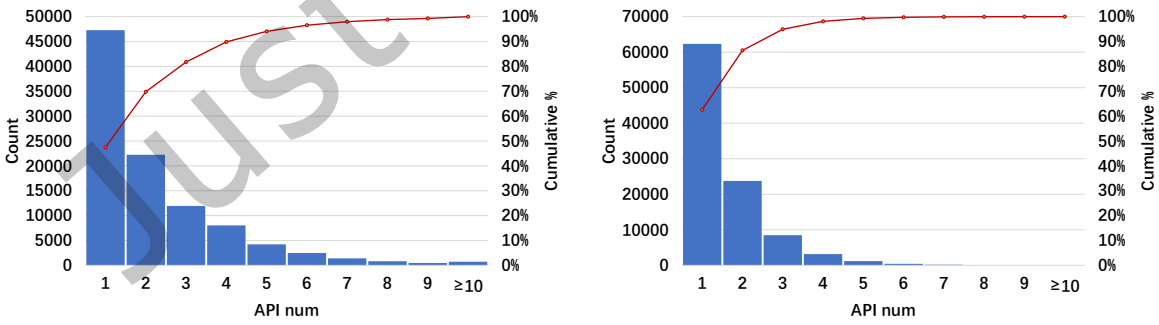
The main contributions of this paper are as follows.

- A retrieval-enhanced LLM-based API recommendation approach, i.e., APIGens, which utilizes the relevant solution samples and API seeds to enhance the LLM model for API sequence generation.
- The effectiveness evaluation for APIGens, which conduct on 265 samples and promising performance was achieved.
- New dataset for API sequence recommendation with queries involving multiple APIs, which provides a new benchmark for this task, and also reminds researchers to pay attention to the multiple APIs recommendation problem.[1]

2 BACKGROUND

2.1 Motivation Study

Previous API recommendation efforts have focused on recommending the best API that can meet queries, but in reality, a query may require multiple APIs to deal with.



(a) Method level

(b) Class level

Fig. 2. Queries being answered with different #API in Stack Overflow

[1]The dataset and source code are publicly available on https://bitbucket.org/mooncakeee/apigens/downloads/

We collect posts from Stack Overflow by using Stack Exchange Data Explorer (detail in Section 4.2) and organize the collected Stack Overflow posts related to API by their API count as demonstrated in Figure 2. We can see that there are more than half of the queries are answered with multiple APIs (short for M-queries) in method level and more then one-third in class level. This also indicates that previous studies evaluated in the dataset with far less M-queries might not be applicable in real-world Q&A scenario. Note that, the reason why the the ratio of M-queries exerts the difference is might because the previous work simplifies the dataset to cope with their approach.

2.2 In-Context Learning

When using LLM to complete tasks, due to the extensive knowledge gained during training. In most cases, we can just use one-shot learning or few shot-learning to let LLM understand the specific task and complete it. In this process, in-context learning (ICL) technology provides an effective prompt scheme.

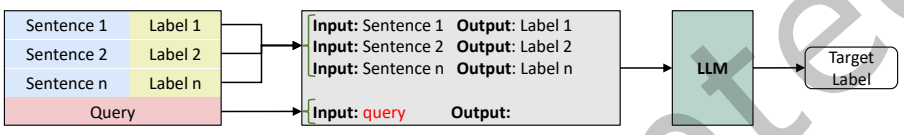


Fig. 3. In-context learning framework

For a task, there is usually an input corresponding to an output. Although we can explain the task requirements to LLM through natural language descriptions, if no corresponding examples are provided, LLM may lack reference. Figure 3 shows the framework of ICL. It can be seen that, ICL will use some samples as part of the prompt template and feed them into LLM, which makes LLM easier to gain the additional knowledge and enables LLM to obtain satisfactory output through analogy. Moreover, this method does not require heavy training of LLM, which is relatively lightweight. [6]

At present, ICL has been proven to be effective and has been used in various tasks [1, 41, 44]. In this paper, in order to obtain better API recommendation sequences, we will also use ICL technology to optimize the generation performance of LLM.

2.3 Retrieve & Re-Rank

In this paper, we will use Retrieve & Re-Rank framework to retrieve the SO posts similar to the given query.

As shown in Figure 4, in this framework, there are two models and the focus of the two models are different. Specifically, the first model needs to be able to get preliminary results quickly, while the second model needs to be able to accurately predict the results. The reason why the second model is not used directly is that the prediction efficiency of the high-performance model is low in general, and when the prediction scope is expanded, the noise received by the model will increase accordingly, which will affect the model performance.

We will use SBERT[31] to implement this framework. In detail, because the search scope of this process is large, an efficient BERT retrieval model (called Bi-Encoder) will be used as the first model to fast retrieval. And then, since the number of candidates after filtered is small, a high accuracy BERT model (called Cross-Encoder) will be used as the second model for re-ranking to refine the candidates.

3 APPROACH

We propose a retrieval-enhanced LLM-based API recommendation approach, i.e., APIGens, which generates API sequences for a natural language query. Its primary idea is to generate the API sequences via the LLM for a given

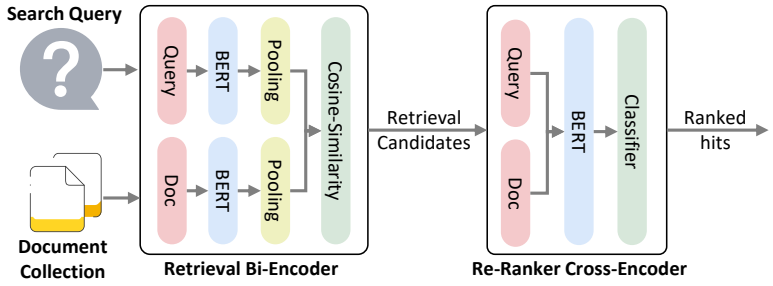


Fig. 4. Retrieve & Re-Rank framework in SBERT

query, and use the the relevant solution samples and API seeds to enhance the generation process to optimize the generated results.

It includes two enhancement modules (Module 1&2) and a basic skeleton module (Module 3), as shown in Figure 5. The basic skeleton module generates API sequence based on a natural language query by utilizing the LLM. Before the basic skeleton begin to generate, there are two enhancement module that will preprocess the input query and boost the prompt feed to LLM.

The first enhancement module will use the historical data to train a retrieval & re-ranking model and when receiving a natural language query, this module will use it to retrieve the similar programming questions from the historical SO database as the candidates for the subsequent process. The second enhancement module will use both the historical data and API document to train a API scorer, and when retrieve the results output by the previous module, this module will calculate the scores between the query and document to re-ranking the retrieved results. After processing through these two modules, multiple reference solutions related to query will be obtained. And then APIGens will use these reference solutions as input samples, and the relevant API of each reference solution will be used as a seed to assist the LLM in generating the API sequence.

3.1 Candidate Solutions Retrieval

This module models the semantic similarity between the query and the historical SO posts, and produces the candidate solutions as input for subsequent modules. The utilization of this module facilitate our approach to pay more attention to some possibly related posts and filter the irrelevant posts from the whole search space.

Since retrieving similar questions from the large SO database is time-consuming, we adopt a retrieve & re-rank framework, which is proven to be efficient for information retrieval from large scale dataset [10, 14, 42].

Specifically, to achieve this goal, APIGens will filter the dissimilar posts from historical data, which is done by utilizing a BERT-based information retrieval model (named $M1$) to retrieve candidate questions that have high similarity with search query at the vector level. The search scope is the SO database $D = [d_1, d_2, ..., d_i, ...]$, and we set the elements d in D as the tuple of question and answered APIs of the SO post and we will use $d.question$ and $d.answer$ to represent them respectively in the rest paper. In order to facilitate subsequent calculations, we will first encode the questions in D to obtain the corresponding vectors.

$$E = [M1.encode(d_1.question), M1.encode(d_2.question), ..., M1.encode(d_i.question), ...] \quad (1)$$

For a given natural language query q, APIGens will use the same way to encode it into a vector.

$$\vec{v} = M1.encode(q) \quad (2)$$

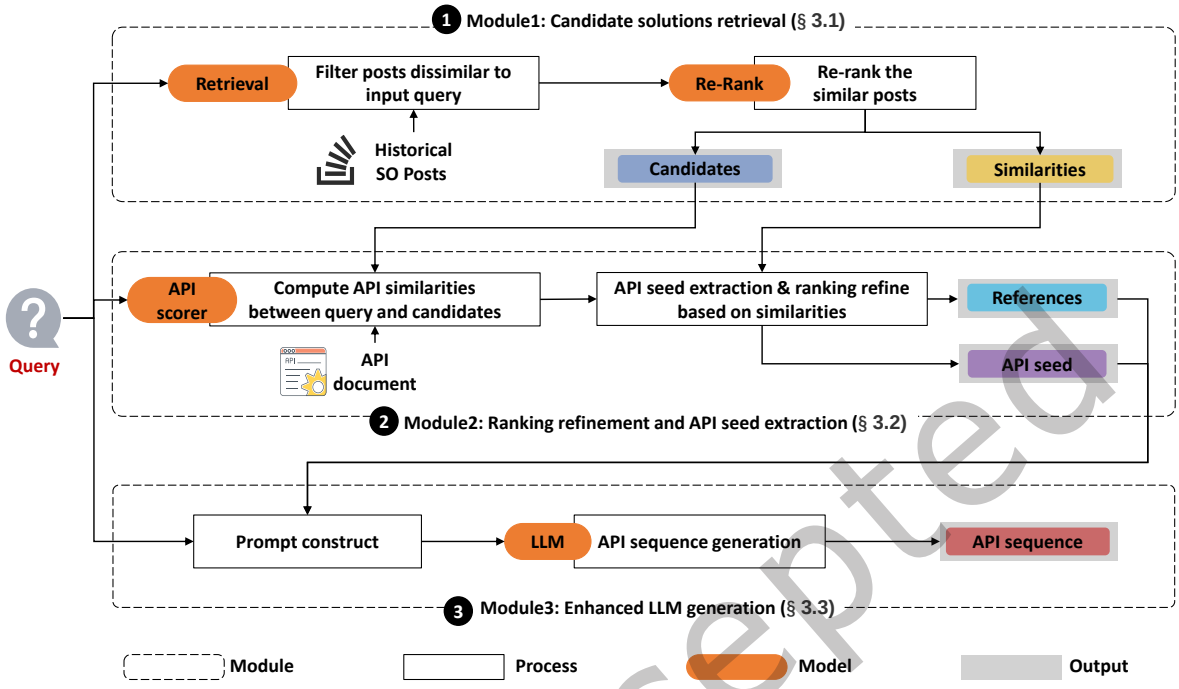


Fig. 5. APIGens overview

And then the similarities U can be calculated by the following formula.

$$U = [\frac{\vec{v} \cdot \vec{e_1}}{|\vec{v}||\vec{e_1}|}, \frac{\vec{v} \cdot \vec{e_2}}{|\vec{v}||\vec{e_2}|}, ..., \frac{\vec{v} \cdot \vec{e_i}}{|\vec{v}||\vec{e_i}|}, ...] \tag{3}$$

Where $\vec{e_i}$ is the i-th element in E. Based on U, we can filter the dissimilar element in D by remain the top (suppose n) posts and get a new set of filtered posts $D^{'}$.

$$\begin{aligned} D^{'} &= [d_{j_1}, d_{j_2}, ..., d_{j_n}] \\ where\ [j_1, j_2, ..., j_n, ...] &= argsort(-U) \end{aligned} \tag{4}$$

In the above formula, the function *argsort* return the indices that would sort an array, and we add a negative sign before U to sort the results in descending order.

After obtaining the filtered posts, another BERT-based model $M2$ will be used to capture the semantic similarity between the query and the questions corresponding to the filtered posts, and determine the ranking based on the relevance scores $U^{'}$ as follow.

$$U^{'} = [M2.predict(q, d_1^{'}.question), M2.predict(q, d_2^{'}.question), ..., M2.predict(q, d_n^{'}.question)] \tag{5}$$

Where the $d_i^{'}$ is the i-th element in $D^{'}$, and different from $M1$ model, the $M2$ model does not encode query and question separately, but requires both of them as inputs. The $D^{'}$ will be re-ranked based on the $U^{'} = [u_1^{'}, u_2^{'}, ..., u_n^{'}]$ to obtain the retrieval results O which contain the similar historical SO posts programming questions and the

corresponding API solutions. After that, the retrieved results and the corresponding scores P will be pass to the next module for further process.

$$\begin{aligned} O = [d^{'}_{j^{'}_1}, d^{'}_{j^{'}_2}, ..., d^{'}_{j^{'}_n}],\ P = [u^{'}_{j^{'}_1}, u^{'}_{j^{'}_2}, ..., u^{'}_{j^{'}_n}] \\ where\ [j^{'}_1, j^{'}_2, ..., j^{'}_n] = argsort(-U^{'}) \end{aligned} \tag{6}$$

In this module, the final output form is several posts retrieved from the Stack Overflow historical data and these posts will be treated as candidates solutions. Each candidate solution includes a *question* and an *answer* (an API sequence). The *question* is the main field used to calculate the similarity with input query, and the *answer* will be used to calculate API scores later. In addition, this module also outputs the similarities corresponding to each candidate solutions for subsequent ranking refinement.

For implementation, we employ SBERT[31], which is introduced in Section 2.3. In SBERT, a Bi-Encoder with greater efficiency is used to roughly filter the irrelevant programming questions, and a Cross-Encoder with higher accuracy is used to re-rank the filtered programming questions. Both encoders are based on BERT [5].

3.2 Ranking Refinement and API Seed Extraction

Through the previous module, APIGens makes full use of historical data, but using only historical data is insufficient. Our goal is to recommend API solutions rather than similar questions, so we should also consider the degree of relevance between API and query.

To achieve this goal, we design another BERT-based Cross-Encoder $M3$ for modeling the semantic similarity between the query and the API documentation to facilitate the capability of predict the correct APIs in the recommendation result.

For the input query and API description, we first connect them through the $[SEP]$ separator, which is a common processing method in the BERT model[5], and feed it into the BERT model to train a similarity prediction model. This input allows the BERT model to carry out the attention mechanism between the two texts, which enables the model to know not only the semantics of each text, but also the semantics of text pair.

In this module, APIGens receives the query q with the corresponding O and P from previous module. The output of this module will be the refined re-ranked candidate solutions and the API seed extracted from them.

Suppose there is a document $C = [c_1, c_2, ..., c_i, ...]$, and we set the elements c in C as the tuple of API name and API comment of the document. We will use $c.name$ and $c.comment$ to represent them respectively in the rest paper. For the given q, APIGens will use $M3$ to calculate the relevance scores R based on top m element in O and P as below.

$$\begin{aligned} R = \frac{1}{2}[p_1 + Score(q, CMT(o_1.answer)), p_2 + Score(q, CMT(o_2.answer)), \\ ..., p_m + Score(q, CMT(o_m.answer))] \\ CMT(answer) = \{c.comment \mid c \in C, c.name \in answer\} \\ Score(q, cmt) = \sum_{a\ in\ cmt} \frac{M3.score(q, a)}{length(cmt)} \end{aligned} \tag{7}$$

Specifically, this step will first obtains the corresponding API comments through the API document, predicts the scores between the query and the API comments using the $M3$ model, calculates the mean value as the API scores, and then calculates the average of API scores and the similarities passed by module 1 as the final relevance scores and refine the ranking based on final relevance scores. The refined re-ranked results $O^{'}$ will be as follow and the *argsort* is the same function as mentioned in Section 3.1:

$$O^{'} = [o_{t_1}, o_{t_2}, ..., o_{t_m}]$$
$$where\ [t_1, t_2, ..., t_n] = argsort(-R) \tag{8}$$

At this point, the approach obtain the refined re-ranked results $O^{'}$ based on API document. The API seeds are the most relevant API in each solution of results. If the API has already been used as a seed before, the sub-relevant API will be extracted, and so on. Finally, this module will output a group of solutions (including *question* and *answer*) for subsequent LLM references, as well as a group of API seeds corresponding to the solutions to enhance the API sequence generation of LLM.

3.3 Enhanced LLM Generation

For the reference solutions and API seeds obtained from the previous two modules, they will be used in this module to generate prompts and input them into the LLM to obtain API sequences. It should be noted that API seeds should not be used independently as they are extracted from reference solutions, and therefore API seeds will accompany the use of reference solutions. While the difference is that reference solutions can be used separately, as their retrieval is independent of API seeds.

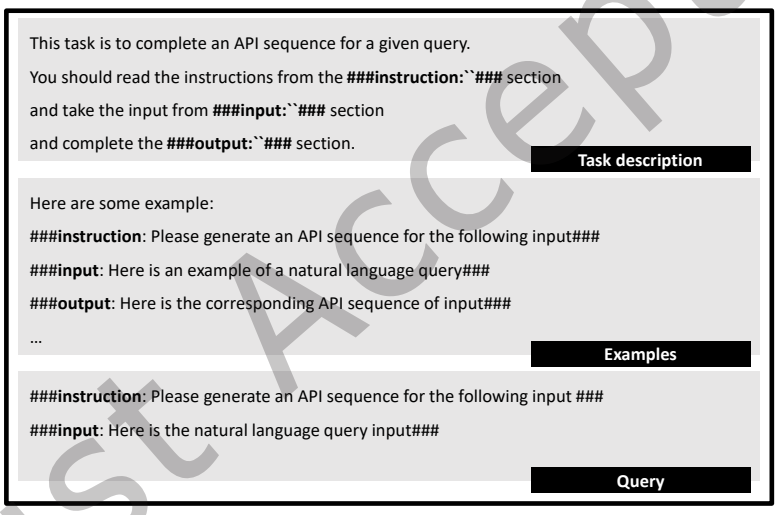


Fig. 6. Prompt use the candidates as samples

Figure 6 shows an reference of a prompt using reference solutions as examples, and it can be seen that the prompt is mainly divided into the following parts:

- **Task description**: Used to inform LLM of the specific task it needs to complete and explain what inputs and outputs it will receive.
- **Examples**: Specific task examples for in-context learning, including instructions, inputs, and outputs.
- **Query**: The sample that need to be answered by LLM, which requires LLM to determine the task form through instructions and make response for the received inputs.

The API seed is used through the example shown in Figure 7. That is, in the prompt examples, additional samples will be added for each reference solutions, which adds an API name to the **instruction** section and asks

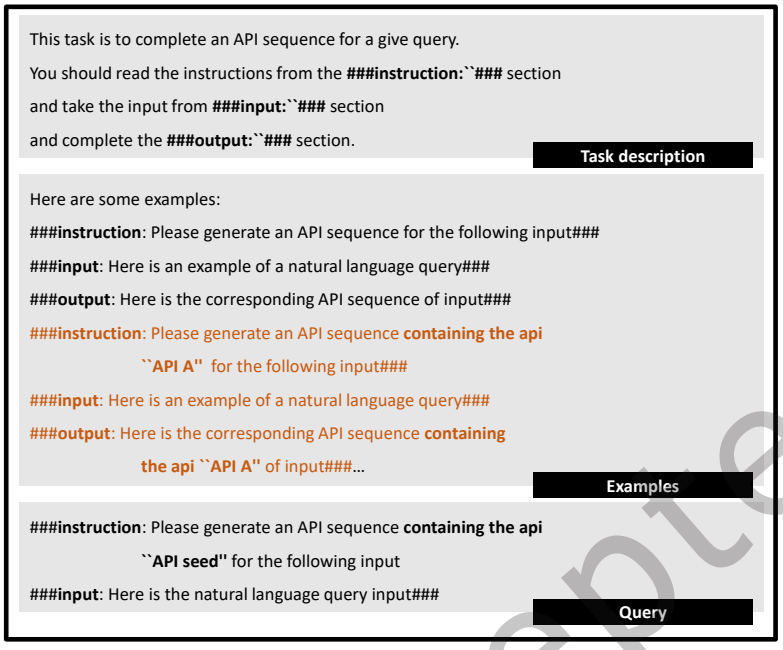


Fig. 7. Prompt use candidates and API seeds

the LLM to generate a target sequence containing that API. When predicting, the API seed can be used to prompt based on the template. It is important to note that in addition to using API seeds, APIGens also retain a seedless result to mitigate the effects of completely incorrect API seeds.

3.4 Model Training

For the first module, we first train the candidate solutions retrieval model, which is for predicting the similarity of a query and SO questions. To build the training data, we pick a question in the training dataset as the query, and calculate the similarity with other SO questions. We measure the jaccard similarity [16] between two API sequences in the answers of these two questions. For each query, we employ contrastive learning [39], which is a strategy of learning through the differences between samples. In detail, for two samples with different relevance with query, the model predicts the relevance scores of query and these two samples respectively, and then we train the model to increase the difference between high relevance samples and low relevance samples and we treat the similarity as the dependent variable(relevance) for model training.

We also need to train the API semantic similarity model, which is for predicting the similarity between a query and an API document. To build the training data, given a question q and its API sequence S, for each API a in S, we treat $<q, a>$ as a positive sample and we randomly choose different APIs not in S as negative samples. Since the positive samples are far less than the negative samples in real scenarios, we follow the setting of previous study [42] to keep the positive samples as 10% when generating the training data.

4 EXPERIMENT DESIGN

4.1 Research Questions

To evaluate our approach, we aim at answering the following research questions.

- RQ1. How effective is APIGens for API sequence recommendation?
- RQ2. How effective is APIGens for API set recommendation?
- RQ3. How do the different modules in APIGens affect the performance?

Note that, APIGens is designed for API sequence generation, while some existing approaches are proposed (or can only) for recommending a set of APIs [14, 42]. To demonstrate the feasibility of APIGens in API set recommendation and its advantages compared with existing approaches, we also treat our generated API sequence as a set of APIs and compare with these approaches. Therefore, we compare APIGens with state-of-the art baselines in both **API sequence recommendation** and **API set recommendation** task in RQ1 and RQ2 respectively. For RQ3, we evaluate the performance of APIGens without the module 2 or module 1&2 as mentioned in Section 3.3, to reveal the benefits of them.

4.2 Dataset

We utilize two datasets for evaluation, i.e., the commonly-used dataset in existing approaches (short for CLEAR's dataset) and newly collected dataset (short for APIGens's dataset).

CLEAR's Dataset: It is the most commonly-used dataset (also used by the state-of-the-art approach [42]). It is first collected by by Huang et al. [14], and some noise is filtered (e.g., SO posts with no API in the answers) following Wei et al. [42]. So we utilize the filtered version provided by [42], which contains 21,479 samples in the training dataset, and 259 samples in the testing dataset.

Furthermore, we notice that more than 90% queries in this dataset are answered with a single API, which cannot fully demonstrate the capability of our proposed approach in recommending API sequence. Therefore, we start out to build a new dataset with queries involving multiple APIs, which can be utilized to demonstrate the performance of API sequence recommendation.

APIGens's Dataset: We obtain all queries and related answers submitted from 2008-08 to 2022-05 from SO website[35, 36] via Stack Exchange Data Explorer[2], then select these queries tagged with *"java"* and having accepted answers following existing studies[42]. This results in 836,122 queries and related answers. After that, we extract the text in <*code*> tags to obtain the code in the accepted answer, then we perform regular matching refer to BIKER's [14] heuristic rules on the extracted text to obtain the API and there are 99,612 samples related to API. Specifically, we identify the API through identifying parentheses and also restore the class of variables through finding the declarations in the code. We further follow BIKER's [14] method which constructs an API name dictionary to recognize the package name of the APIs, and retained posts that involve multiple API in class level. Eventually collect 37,282 samples with API sequence. For this part of the dataset, we use LLM to clean it up, removing questions that LLM believe are not programming tasks or needed clarification, and obtain 15,606 samples. Specifically, in the LLM cleanup process, for each sample, we ask the LLM whether it is a programming task (asking it to answer yes or no) and give it 10 examples as references. We keep the samples that LLM considers to be programming task. We also applied the same procedure for whether the samples needed clarification, and keep the samples that the LLM do not think needed clarification.

For the testing set, we refer the BIKER's criterion to generate, i.e., filter out the samples that do not ask a coding question or have an unclear description of the purpose, etc., and finally result in 265 samples.

[2]https://data.stackexchange.com/

Note that, we utilize CLEAR's dataset for evaluating the performance of API set recommendation since the recommended APIs in this dataset are unordered, and we utilize APIGens's dataset for evaluating the performance of API sequence recommendation.

4.3 Baselines

DeepAPI[9]: the commonly-used approach for API sequence generation with seq2seq model. It models the API recommendation as the machine translation task, and utilizes the RNN model to translate the natural language query to an API sequence[12, 33].

DGAS[40]: the state-of-the-art API sequence search approach with attention network. It uses inner attention mechanism and cross attention mechanism to calculate similarity based on three input, i.e., API, document, and query. Firstly, DGAS will calculate the attention encoding for the three separately, and then the combination of these encoding will be used to further calculate the final representation of query, API and document. Finally, the cosine similarity will be calculated to search the similar API sequences.

BIKER[14]: the commonly-used approach for API set recommendation, which can only recommend a set of APIs, rather than the API sequence. It first recommends related queries to retrieve the candidate APIs, and then re-ranks the candidates based on the similarity between the query and API documentation with TF-IDF and word2vec[23] techniques.

CLEAR[42]: the state-of-the-art approach for API set recommendation. It retrieves the relevant questions from the dataset with BERT-based embedding model and contrastive learning technique[39], and recommends the corresponding APIs from the retrieved questions.

To summarize, we utilize DeepAPI and DGAS as baselines for API sequence recommendation on APIGens's dataset, and utilize BIKER and CLEAR as baselines for API set recommendation on CLEAR's dataset.

4.4 Evaluation Metrics

To evaluate the performance of our approach on API sequence generation and API set recommendation, we utilize different metrics following existing studies [9, 14, 42]. Specifically, we employ ROUGE, which is usually used in sequence generation task, for API sequence recommendation, and utilize Recall@K, MAP, and MRR for API set recommendation. Since APIGens can obtain top-K sequences, we will calculate the maximum ROUGE of recommended top-K sequences, and record them as ROUGE@K.

ROUGE (Recall-Oriented Understudy for Gisting Evaluation) [18] compares an automatically generated sequence with a set of reference sequences, and calculates the corresponding score to measure the similarity between the automatically generated sequence and the reference sequences. In this study, we adopt ROUGE-L as a metric.

Recall@K [48] measures the proportion of the correct API recommended in the top-K recommendations among all ground truth APIs. It evaluates the model's ability in finding all correct APIs.

MAP (Mean Average Precision) [34] is defined as the mean of the Average Precision (*AP*) values obtained for all the evaluation queries. The *AP* is the sum of the product of recall change and corresponding precision at every position in the recommended result.

MRR (Mean Reciprocal Rank) [28] is a general metric of information retrieval algorithm evaluation, which can examine the quality of recommendation results in ranking order.

4.5 Experimental Settings

For RQ1, we run our approach and baselines on the APIGens's dataset, and obtain their performance in terms of the method-level sequences recommendation and class-level sequences recommendation.

For RQ2, similar to RQ1, we will use the CLEAR's dataset for experiments, running our approach and baselines on the dataset to obtain their performance on both level, and preserve the experimental settings of CLEAR if possible. For the recommendation results of APIGens, we rank the APIs in order of their appearance in the ranked sequences and only consider the first appearance.

For RQ3, we remove the module 2 or module 1&2, and obtain the recommendation performance to demonstrate the benefits of them. In addition, when the second module is removed, the candidate solutions will be directly used as the prompt examples and we will set the parameters of LLM to enable it to return multiple responses because there will be no API seeds. Since the second module needs to be based on the output of the first module, the first module will not be removed independently and when the both modules are removed, we will use random samples as the prompt examples.

For the retrieval information source used by APIGens, we adopt the training set used in the corresponding RQ to ensure consistency with the usage scenario and we use the API documents collected in BIKER in our experiments. For the LLM, we adopt GPT-3.5-turbo-1106 [27] to complete subsequent experiments.

5 RESULTS

5.1 Answering RQ1 API Sequence Recommendation

Table 2. Performance comparison of different approaches on API sequence dataset (RQ1)

API Level	Method-Level					Class-Level				
Approach	OUR	DeepAPI		DGAS		OUR	DeepAPI		DGAS	
V(%) & I*	Val.	Val.	Imp.	Val.	Imp.	Val.	Val.	Imp.	Val.	Imp.
ROUGE@1	**24.03**	15.24	57.68%	12.39	93.95%	**40.43**	29.71	36.08%	24.55	64.68%
ROUGE@3	**38.03**	18.82	102.07%	16.88	125.30%	**53.47**	37.42	42.89%	28.85	85.34%
ROUGE@5	**42.13**	20.56	104.91%	19.28	118.52%	**58.13**	41.51	40.04%	31.03	87.33%
ROUGE@10	**48.41**	23.43	106.62%	23.55	105.56%	**64.78**	43.81	47.87%	34.02	90.42%

*Value and Improvement, underline: $p < 0.05$

Table 2 presents the ROUGE values of APIGens and two baselines, both at the method-level and the class-level. Since method-level and class-level demonstrate similar trend, the following analysis focuses on the method-level performance, which is more-widely used. In addition, we also presents the Wilcoxon signed-rank test [43] results. The underlined parts in the experimental results indicate that the performance is significantly lower than the corresponding performance of our approach.

APIGens can outperform the baselines in all metrics. It can achieve 24.03% ROUGE@1, 38.03% ROUGE@3, and 42.13% ROUGE@5, and 48.41% ROUGE@10 respectively. The relatively high ROUGE indicates that our approach can generate the correct API sequences. For the general generative models, latest researches [4, 17, 32] achieved around 40% ROUGE-L, and relevant AI company documents [26] also mention that a ROUGE-L exceeding 45% can be considered a high level. Considering the task is different, our task target sequence is shorter, which lead to worse fault tolerance, so it can be considered that the results of our approach are satisfying. Taken in this sense, APIGens achieves promising performance.

Compared with DeepAPI, APIGens improves it by 57.68% in ROUGE@1, and 106.62% in ROUGE@10. When compared with DGAS, the performance improvement is respectively 93.95% in ROUGE@1, and 105.56% in ROUGE@10. This is because we adopt the state-of-the-art generation model, i.e., large language model, and incorporate both the relevance samples and extracted API seeds. DeepAPI only considers the direct translation of the input query into API sequence, and does not utilize any other information. While for DGAS, it only use the

API document data, and without the process for filtering irrelevant historical data. Thus their performance are relatively poorer.

APIGens can achieve the ROUGE@10 of 48.41% on API sequence recommendation, which outperforms the state-of-the-art baselines in a large margin.

5.2 Answering RQ2 API Set Recommendation

Table 3. Performance comparison of different approaches on API set dataset (RQ2)

API Level	Method-Level					Class-Level				
Approach	OUR	BIKER		CLEAR		OUR	BIKER		CLEAR	
V(%) & I*	Val.	Val.	Imp.	Val.	Imp.	Val.	Val.	Imp.	Val.	Imp.
Recall@1	**70.46**	45.95	53.36%	59.46	18.51%	**81.66**	61.97	31.78%	77.41	5.49%
Recall@3	**90.35**	69.31	30.36%	80.69	11.96%	**94.59**	84.17	12.39%	93.05	1.66%
Recall@5	**93.82**	79.34	18.25%	86.10	8.97%	**96.33**	91.89	4.83%	95.56	0.81%
Recall@10	**96.33**	89.19	8.01%	91.31	5.50%	**97.49**	95.75	1.81%	96.33	1.20%
MAP	**82.61**	61.23	34.92%	72.56	13.86%	**88.95**	74.67	19.13%	85.81	3.66%
MRR	**82.97**	62.38	33.02%	72.96	13.72%	**90.85**	75.29	20.67%	87.64	3.66%

*Value and Improvement, underline: $p < 0.05$

Table 3 presents the Recall@K, MAP and MRR values of each approach, both at method-level and class-level. As previous section, we focus on the method-level performance.

We can see that, among the approaches, APIGens can outperform others in all metrics. Our approach can achieve 96.33%, 82.61%, 82.97% on Recall@10, MAP and MRR respectively, which indicates that our approach can find at least one correct API in the top 10 recommendation in 96.33% cases.

Compared with BIKER, our approach improves by 8.01%, 34.92% and 34.02% in the three metrics of Recall@10, MAP and MRR respectively, and by 5.50%, 13.86% and 13.72% compared with CLEAR. The higher Recall@K means that compared with other approaches, our approach can find more correct APIs when the number of retrieves is fixed, and the higher MAP and MRR indicate that our approach can rank the correct API higher than other approaches. This might because the BIKER utilizes the simple similarity metrics, i.e., word2vec and TFIDF, for retrieving the related posts and recommending APIs, which is less accurate. Besides, CLEAR utilizes the BERT-based retrieve and re-rank framework for API recommendation, which is superior than BIKER. Nevertheless, we design our approach to incorporate the BERT and LLM for better understanding the query and the APIs, which achieves the highest performance.

In addition, we also observe that with the number of candidate recommendations increases, the performance between our approach and the baselines becomes closer. Yet in the API recommendation scenario, one would pay more attention to the recommendation ranked in the top, and might not have the patient to glance the tenth recommendations, which further indicates the superior of our approach.

For class level, we can see that the performance difference between our approach and CLEAR is small, which may be due to the less variations of API set recommendations at the class level, making it easy to retrieve answers, so the improvement between the two is not significant.

APIGens can achieve the MAP value of 82.61% on API set recommendation, which outperforms the state-of-the-art baselines in a large margin.

5.3 Answering RQ3 Module Contribution

Table 4. Performance comparison at different modules (RQ3)

Level	Models	ROUGE@1	ROUGE@3	ROUGE@5	ROUGE@10
Method	APIGens complete	**24.03**↑8.93%	**38.03**↑17.12%	**42.13**↑17.48%	**48.41**↑28.07%
	APIGens w/o API module	22.06↑28.03%	32.47↑17.14%	35.86↑17.77%	37.80↑16.52%
	APIGens w/o both modules	17.23	27.72	30.45	32.44
Class	APIGens complete	**40.43**↑0.20%	**53.47**↑3.70%	**58.13**↑8.41%	**64.78**↑16.97%
	APIGens w/o API module	40.35↑16.05%	51.56↑13.00%	53.62↑11.31%	55.38↑11.90%
	APIGens w/o both modules	34.77	45.63	30.45	49.49

underline: $p < 0.05$. The improvement of the target model is based on the model in the next row, i.e,
APIGens complete VS APIGens w/o API module and
APIGens w/o API module VS APIGens w/o both modules

We incorporate both the relevance samples and extracted API seeds in the basic LLM to boost the performance. Table 4 shows the performance of removing the module 2 and both modules as mentioned in Section 4.5. We can find that using all modules, i.e., the proposed approach, has the highest performance and can achieve 48.41% on ROUGE@10 at method level and 67.78% at class level, which improves ROUGE@10 by 28.07% at method level and 16.97% at class level compared to the approach without API module.

In addition, it can be seen that the improvement at the class level is less than the improvement at the method level. This may be because the class level recommendation greatly reduces the scope of API retrieval, making retrieval easier.

From the results, it can be seen that the contribution of module 1 is mainly in recommending more relevant results. However, due to the lack of API seeds, its diversity is limited and it is prone to encountering local optima. Therefore, as k increases, the improvement rate will not increase. Module 2 receives the output of module 1 and extracts relevant APIs as API seeds. As these APIs are related to the query, it greatly increases the reasonable generation scope, thereby increasing the diversity of results and reducing the possibility of local optima. Therefore, it can be seen that as k increases, the improvement rate also continues to increase, because it includes more related APIs.

The both module contributes to the API sequence recommendation. Module 1 mainly focus on the relevance of results and module 2 mainly focus on diversity of results.

6 DISCUSSION

6.1 Complementary of Retrieval and Generation

From our approach design in Figure 5, it can be seen that we use the retrieval re-ranking model and LLM, which involve two different features of our approach, namely the retrieval feature of the first module and the generation feature of the third module. These two features are suitable for different scenarios, and retrieval feature can achieve higher effectiveness of solutions because the recommend solutions have been confirmed by developers in the past. Therefore, this is more suitable for recommending solutions to common programming questions,

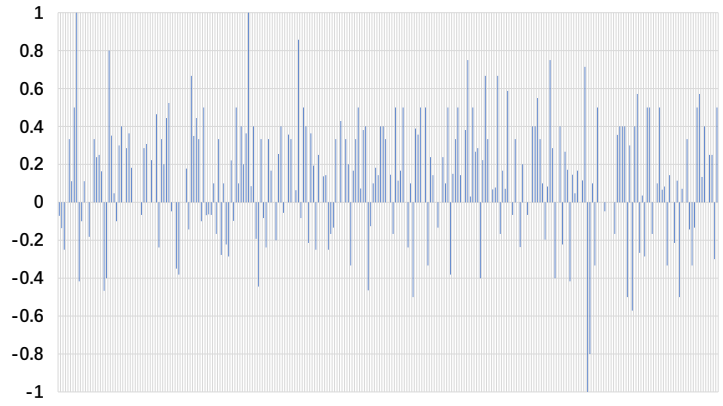


Fig. 8. Performance differences between retrieval solutions and generative solutions

which highly beneficial for junior developers because they are more likely to raise questions about some basic programming knowledge. On the other hand, for the generation feature of LLM, it is more creative, that is, its solution is written by LLM based on programming questions combined with its own knowledge, so this is more suitable for some rare programming questions, and therefore more beneficial for senior developers.

From Table 4, it can be seen that the generation performance of the LLM is not promising without relevant references and API seeds. However, as mentioned earlier, this does not mean that it can be completely dependent on retrieval feature to complete the API recommendation task. Figure 8 shows the difference in ROUGE@10 between the API sequence results generated using only LLM and API sequence in the candidate solutions obtained from module 1 (as mentioned in Section 3.1). The horizontal axis represents the test samples, and the vertical axis represents the difference between the two recommended results on that sample. The upper half of the horizontal axis indicates that the candidate solution is superior to the generated solution, while the lower half indicates that the generated solution is superior to the candidate solution. The length of the bar represents the difference between the two results. It can be seen that not all candidate solutions have higher ROUGE@10 on all samples, and for some samples, the generation solutions is superior to the candidate solutions, which represents the complementarity between retrieval feature and generation feature.

6.2 Impact of Pre-trained Knowledge

Artificial intelligence technology has made significant progress. In recent years, the emergence of large language models bring this field into a new era, expanding artificial intelligence from artificial narrow intelligence (ANI) to artificial general intelligence (AGI), unifying specific problem paradigms into unified paradigms. Although the structure and volume of artificial intelligence models have become increasingly sophisticated, what remains unchanged is that researchers are trying to incorporate knowledge into the weights of the models to obtain pre-trained models. Afterwards, we complete our task by fine-tuning or directly using these pre-trained models.

In this paper, we used ChatGPT as the LLM for sequence generation in our approach, and in addition to ChatGPT, there are many open-source LLMs that can be used as part of our approach. For example, the Llama model published by Facebook[38] and the Gemma model published by Google[22]. Therefore, in this section, we will use these models for API sequence generation to evaluate performance differences.

Table 5 shows the performance of using Llama3-8B and Gemma-7B model, and we can see that the performance of both models is not as good as ChatGPT, which is easy to understand because ChatGPT has a much larger number

Table 5. Performance comparison of different open source LLM

API Level	Method-Level					Class-Level				
Approach	OUR	Llama3-8B		Gemma-7B		OUR	Llama3-8B		Gemma-7B	
V(%) & I*	Val.	Val.	Imp.	Val.	Imp.	Val.	Val.	Imp.	Val.	Imp.
ROUGE@1	**24.03**	21.20	13.35%	19.25	24.83%	**40.43**	33.93	19.16%	31.80	27.14%
ROUGE@3	**38.03**	33.58	13.25%	31.33	21.39%	**53.47**	47.58	12.38%	47.13	13.45%
ROUGE@5	**42.13**	37.18	13.31%	35.49	18.71%	**58.13**	51.77	12.29%	53.36	8.94%
ROUGE@10	**48.41**	43.27	11.88%	39.97	21.12%	**64.78**	58.28	11.15%	61.12	5.99%

*Value and Improvement, underline: $p < 0.05$

of parameters than these two models. However, these two models have also shown promising performance, so they can also be used for some low-cost solutions. Due to limitations in computing resources, we did not conduct experiments on larger LLM and we will consider it as a future work.

Both of the above models belong to AGI models. In fact, before AGI models became popular, researchers mainly relied on some ANI models for fine-tuning to form solutions. These ANI models are often trained based on domain specific knowledge. Although they do not have the ability to solve general problems, they often have impressive performance in domain task. For example, CodeBert[8] based on BERT and CodeTrans[7] based on T5[29] are widely used in code related tasks, such as code document generation. In addition, Elnaggar et al. and Martin et al. also used these models to generate API sequence[7, 20]. We also fine-tune these two models to evaluate their performance and explore the impact of pre-training knowledge.

Table 6. Performance comparison of different pre-trained models

API Level	Method-Level					Class-Level				
Approach	OUR	CodeBert		CodeTrans		OUR	CodeBert		CodeTrans	
V(%) & I*	Val.	Val.	Imp.	Val.	Imp.	Val.	Val.	Imp.	Val.	Imp.
ROUGE@1	**24.03**	17.32	38.74%	16.74	43.55%	**40.43**	33.48	20.76%	29.87	35.35%
ROUGE@3	**38.03**	23.93	58.92%	21.89	73.73%	**53.47**	46.49	15.01%	41.55	28.69%
ROUGE@5	**42.13**	27.40	53.76%	24.56	71.54%	**58.13**	51.28	13.36%	46.39	25.31%
ROUGE@10	**48.41**	32.36	49.60%	30.85	56.92%	**64.78**	58.06	11.57%	54.08	19.79%

*Value and Improvement, underline: $p < 0.05$

As shown in Table 6, we can see that the performance of both models is lower than our approach, but compared to DeepAPI and DGAS, they have large improvements, indicating that pre training knowledge can play an important role in the task of this paper.

Overall, from the above experimental results, it can be seen that compared to traditional methods, pre-training knowledge has a significant impact, which is also in line with the current development trend of artificial intelligence, which constructs intelligence with massive knowledge.

6.3 Threats to validity

The first threat comes from our implementation of the baseline approach and collection of the dataset. To mitigate this threat, we reuse the replication packages provided by the original paper to ensure its correctness. For DGAS, which did not provide the code, we implemented it as described in the original paper, since it did not provide the

static Word2vec encoding, we used the pre-trained model provided by Google[3], so the encoding dimensions also set according to that pre-trained model. For our collected API sequence dataset, we follow existing studies and refer to the heuristic rules to filter the datasets. The second threat comes from our selection of evaluation metrics. Inappropriate evaluation metrics will lead to misleading experimental results. In order to reduce this threat, the evaluation metrics we selected are commonly-used in the research of sequence generation or recommendation task.

7 RELATED WORK

Recommending APIs from the natural language query is widely studies in previous[9, 14, 42]. There are also some researches that provide auxiliary tools to optimize the API retrieval process. McMillan et al.[21] proposed the Portfolio code search system, which can provide visual API related function search results for programmers. Thung et al.[37] applied the API recommendation approach to the feature request (FR) scenario and automatically recommended API by querying historical FR database and API documentation. Rahman et al.[30] proposed the RACK tool, which constructs a top-K API association set based on a large amount of data from Stack Overflow. However, this approach can only recommend classes and therefore does not serve as our baseline. Yuan et al.[49] studied API recommendation for event-driven programming frameworks such as the Android framework, built the LibraryGuru recommendation library, which is a database containing Android-related APIs. Xie et al.[47] conducted a large-scale empirical research, and extracted 87 classes of 356 functional verbs and 523 phrase patterns. Based on these verbs, programmers can build an appropriate query. These studies are mainly used in the specific scenarios or need specific configuration, so they are not considered in this paper. Huang et al.[13] designed a knowledge graph based multi-round human-machine interaction to facilitate the query clarification and recommend an API based on the user's answers. Apart from the natural language based API recommendation approach, some studies focused on the context aware API recommendation [2, 3, 11, 15, 19, 24, 25, 46], which recommends the suitable APIs according to the programming context. The input of theirs are different from this work, therefore we do not utilize them as the baselines.

8 CONCLUSION

In this paper, we propose APIGens, a retrieval-enhanced large language model based API recommendation approach to recommend API sequences for users by incorporating the historical data and API documents. Our experimental results confirm the effectiveness of APIGens for API recommendation. Furthermore, we also discuss the complementary of retrieval and generation for recommending the API sequence and prove that both of them are useful for the API sequence recommendation task. In the future, we will continue to develop tools, optimize the user experience, and provide more convenient API sequence recommendation services.

ACKNOWLEDGMENTS

This work was supported by the National Natural Science Foundation of China Grant No.62232016, No.62072442, No.62272445 and No.62402483, Youth Innovation Promotion Association Chinese Academy of Sciences, Basic Research Program of ISCAS Grant No. ISCAS-JCZD-202304, and Major Program of ISCAS Grant No. ISCAS-ZD-202302.

REFERENCES

[1] Amir Bar, Yossi Gandelsman, Trevor Darrell, Amir Globerson, and Alexei A. Efros. 2022. Visual Prompting via Image Inpainting. In *NeurIPS*. http://papers.nips.cc/paper_files/paper/2022/hash/9f09f316a3eaf59d9ced5ffaefe97e0f-Abstract-Conference.html

[3]https://code.google.com/archive/p/word2vec

[2] Chi Chen, Xin Peng, Bihuan Chen, Jun Sun, Zhenchang Xing, Xin Wang, and Wenyun Zhao. 2022. "More Than Deep Learning": post-processing for API sequence recommendation. *Empir. Softw. Eng.* 27, 1 (2022), 15. https://doi.org/10.1007/s10664-021-10040-2

[3] Chi Chen, Xin Peng, Zhenchang Xing, Jun Sun, Xin Wang, Yifan Zhao, and Wenyun Zhao. 2021. Holistic combination of structural and textual code information for context based api recommendation. *IEEE Transactions on Software Engineering* (2021).

[4] Lu Chen, Ruqing Zhang, Wei Huang, Wei Chen, Jiafeng Guo, and Xueqi Cheng. 2023. Inducing Causal Structure for Abstractive Text Summarization. In *Proceedings of the 32nd ACM International Conference on Information and Knowledge Management, CIKM 2023, Birmingham, United Kingdom, October 21-25, 2023*, Ingo Frommholz, Frank Hopfgartner, Mark Lee, Michael Oakes, Mounia Lalmas, Min Zhang, and Rodrygo L. T. Santos (Eds.). ACM, 213–223. https://doi.org/10.1145/3583780.3614934

[5] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, Jill Burstein, Christy Doran, and Thamar Solorio (Eds.). Association for Computational Linguistics, 4171–4186. https://doi.org/10.18653/v1/n19-1423

[6] Qingxiu Dong, Lei Li, Damai Dai, Ce Zheng, Zhiyong Wu, Baobao Chang, Xu Sun, Jingjing Xu, Lei Li, and Zhifang Sui. 2023. A Survey for In-context Learning. *CoRR* abs/2301.00234 (2023). https://doi.org/10.48550/ARXIV.2301.00234 arXiv:2301.00234

[7] Ahmed Elnaggar, Wei Ding, Llion Jones, Tom Gibbs, Tamas Feher, Christoph Angerer, Silvia Severini, Florian Matthes, and Burkhard Rost. 2021. CodeTrans: Towards Cracking the Language of Silicone's Code Through Self-Supervised Deep Learning and High Performance Computing. *CoRR* abs/2104.02443 (2021). arXiv:2104.02443 https://arxiv.org/abs/2104.02443

[8] Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, and Ming Zhou. 2020. CodeBERT: A Pre-Trained Model for Programming and Natural Languages. In *Findings of the Association for Computational Linguistics: EMNLP 2020, Online Event, 16-20 November 2020 (Findings of ACL, Vol. EMNLP 2020)*, Trevor Cohn, Yulan He, and Yang Liu (Eds.). Association for Computational Linguistics, 1536–1547. https://doi.org/10.18653/V1/2020.FINDINGS-EMNLP.139

[9] Xiaodong Gu, Hongyu Zhang, Dongmei Zhang, and Sunghun Kim. 2016. Deep API learning. In *Proceedings of the 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering, FSE 2016, Seattle, WA, USA, November 13-18, 2016*, Thomas Zimmermann, Jane Cleland-Huang, and Zhendong Su (Eds.). ACM, 631–642. https://doi.org/10.1145/2950290.2950334

[10] Vishal Gupta, Manoj Chinnakotla, and Manish Shrivastava. 2018. Retrieve and Re-rank: A Simple and Effective IR Approach to Simple Question Answering over Knowledge Graphs. In *Proceedings of the First Workshop on Fact Extraction and VERification (FEVER)*. Association for Computational Linguistics, Brussels, Belgium, 22–27. https://doi.org/10.18653/v1/W18-5504

[11] Xincheng He, Lei Xu, Xiangyu Zhang, Rui Hao, Yang Feng, and Baowen Xu. 2021. PyART: Python API Recommendation in Real-Time. In *43rd IEEE/ACM International Conference on Software Engineering, ICSE 2021, Madrid, Spain, 22-30 May 2021*. IEEE, 1634–1645. https://doi.org/10.1109/ICSE43902.2021.00145

[12] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long Short-Term Memory. *Neural Comput.* 9, 8 (1997), 1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735

[13] Qing Huang, Zhenyu Wan, Zhenchang Xing, Changjing Wang, Jieshan Chen, Xiwei Xu, and Qinghua Lu. 2023. Let's Chat to Find the APIs: Connecting Human, LLM and Knowledge Graph through AI Chain. In *38th IEEE/ACM International Conference on Automated Software Engineering, ASE 2023, Luxembourg, September 11-15, 2023*. IEEE, 471–483. https://doi.org/10.1109/ASE56229.2023.00075

[14] Qiao Huang, Xin Xia, Zhenchang Xing, David Lo, and Xinyu Wang. 2018. API method recommendation without worrying about the task-API knowledge gap. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering, ASE 2018, Montpellier, France, September 3-7, 2018*, Marianne Huchard, Christian Kästner, and Gordon Fraser (Eds.). ACM, 293–304. https://doi.org/10.1145/3238147.3238191

[15] Ivana Clairine Irsan, Ting Zhang, Ferdian Thung, Kisub Kim, and David Lo. 2023. Multi-Modal API Recommendation. In *IEEE International Conference on Software Analysis, Evolution and Reengineering, SANER 2023, Taipa, Macao, March 21-24, 2023*, Tao Zhang, Xin Xia, and Nicole Novielli (Eds.). IEEE, 272–283. https://doi.org/10.1109/SANER56733.2023.00034

[16] Paul Jaccard. 1912. The distribution of the flora in the alpine zone. 1. *New phytologist* 11, 2 (1912), 37–50.

[17] Khang Nhut Lam, Thieu Gia Doan, Khang Thua Pham, and Jugal Kalita. 2023. Abstractive Text Summarization Using the BRIO Training Paradigm. In *Findings of the Association for Computational Linguistics: ACL 2023, Toronto, Canada, July 9-14, 2023*, Anna Rogers, Jordan L. Boyd-Graber, and Naoaki Okazaki (Eds.). Association for Computational Linguistics, 92–99. https://doi.org/10.18653/V1/2023.FINDINGS-ACL.7

[18] Chin-Yew Lin. 2004. ROUGE: A Package for Automatic Evaluation of Summaries. In *Text Summarization Branches Out*. Association for Computational Linguistics, Barcelona, Spain, 74–81. https://aclanthology.org/W04-1013

[19] Xiaoyu Liu, LiGuo Huang, and Vincent Ng. 2018. Effective API recommendation without historical software repositories. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering, ASE 2018, Montpellier, France, September 3-7, 2018*, Marianne Huchard, Christian Kästner, and Gordon Fraser (Eds.). ACM, 282–292. https://doi.org/10.1145/3238147.3238216

[20] James Martin and Jin L. C. Guo. 2022. Deep API learning revisited. In *Proceedings of the 30th IEEE/ACM International Conference on Program Comprehension, ICPC 2022, Virtual Event, May 16-17, 2022*, Ayushi Rastogi, Rosalia Tufano, Gabriele Bavota, Venera Arnaoudova,

and Sonia Haiduc (Eds.). ACM, 321–330. https://doi.org/10.1145/3524610.3527872

[21] Collin McMillan, Mark Grechanik, Denys Poshyvanyk, Qing Xie, and Chen Fu. 2011. Portfolio: finding relevant functions and their usage. In *Proceedings of the 33rd International Conference on Software Engineering, ICSE 2011, Waikiki, Honolulu , HI, USA, May 21-28, 2011*, Richard N. Taylor, Harald C. Gall, and Nenad Medvidovic (Eds.). ACM, 111–120. https://doi.org/10.1145/1985793.1985809

[22] Thomas Mesnard, Cassidy Hardin, Robert Dadashi, Surya Bhupatiraju, Shreya Pathak, Laurent Sifre, Morgane Rivière, Mihir Sanjay Kale, Juliette Love, Pouya Tafti, Léonard Hussenot, Aakanksha Chowdhery, Adam Roberts, Aditya Barua, Alex Botev, Alex Castro-Ros, Ambrose Slone, Amélie Héliou, Andrea Tacchetti, Anna Bulanova, Antonia Paterson, Beth Tsai, Bobak Shahriari, Charline Le Lan, Christopher A. Choquette-Choo, Clément Crepy, Daniel Cer, Daphne Ippolito, David Reid, Elena Buchatskaya, Eric Ni, Eric Noland, Geng Yan, George Tucker, George-Christian Muraru, Grigory Rozhdestvenskiy, Henryk Michalewski, Ian Tenney, Ivan Grishchenko, Jacob Austin, James Keeling, Jane Labanowski, Jean-Baptiste Lespiau, Jeff Stanway, Jenny Brennan, Jeremy Chen, Johan Ferret, Justin Chiu, and et al. 2024. Gemma: Open Models Based on Gemini Research and Technology. *CoRR* abs/2403.08295 (2024). https://doi.org/10.48550/ARXIV.2403.08295 arXiv:2403.08295

[23] Tomás Mikolov, Ilya Sutskever, Kai Chen, Gregory S. Corrado, and Jeffrey Dean. 2013. Distributed Representations of Words and Phrases and their Compositionality. In *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States*, Christopher J. C. Burges, Léon Bottou, Zoubin Ghahramani, and Kilian Q. Weinberger (Eds.). 3111–3119. https://proceedings.neurips.cc/paper/2013/hash/9aa42b31882ec039965f3c4923ce901b-Abstract.html

[24] Anh Tuan Nguyen, Michael Hilton, Mihai Codoban, Hoan Anh Nguyen, Lily Mast, Eli Rademacher, Tien N. Nguyen, and Danny Dig. 2016. API code recommendation using statistical learning from fine-grained changes. In *Proceedings of the 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering, FSE 2016, Seattle, WA, USA, November 13-18, 2016*, Thomas Zimmermann, Jane Cleland-Huang, and Zhendong Su (Eds.). ACM, 511–522. https://doi.org/10.1145/2950290.2950333

[25] Phuong T Nguyen, Juri Di Rocco, Claudio Di Sipio, Davide Di Ruscio, and Massimiliano Di Penta. 2021. Recommending api function calls and code snippets to support software development. *IEEE Transactions on Software Engineering* (2021).

[26] OneAI. 2023. Rouge metrics for Summary & Headline. https://docs.oneai.com/docs/rouge-metrics-for-summary-headline.

[27] OpenAI. 2023. New models and developer products announced at DevDay. https://openai.com/index/new-models-and-developer-products-announced-at-devday/.

[28] Dragomir R. Radev, Hong Qi, Harris Wu, and Weiguo Fan. 2002. Evaluating Web-based Question Answering Systems. In *Proceedings of the Third International Conference on Language Resources and Evaluation, LREC 2002, May 29-31, 2002, Las Palmas, Canary Islands, Spain*. European Language Resources Association. http://www.lrec-conf.org/proceedings/lrec2002/sumarios/301.htm

[29] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. *J. Mach. Learn. Res.* 21 (2020), 140:1–140:67. http://jmlr.org/papers/v21/20-074.html

[30] Mohammad Masudur Rahman, Chanchal Kumar Roy, and David Lo. 2016. RACK: Automatic API Recommendation Using Crowdsourced Knowledge. In *IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering, SANER 2016, Suita, Osaka, Japan, March 14-18, 2016 - Volume 1*. IEEE Computer Society, 349–359. https://doi.org/10.1109/SANER.2016.80

[31] Nils Reimers and Iryna Gurevych. 2019. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics. https://arxiv.org/abs/1908.10084

[32] Paul Roit, Johan Ferret, Lior Shani, Roee Aharoni, Geoffrey Cideron, Robert Dadashi, Matthieu Geist, Sertan Girgin, Léonard Hussenot, Orgad Keller, Nikola Momchev, Sabela Ramos Garea, Piotr Stanczyk, Nino Vieillard, Olivier Bachem, Gal Elidan, Avinatan Hassidim, Olivier Pietquin, and Idan Szpektor. 2023. Factually Consistent Summarization via Reinforcement Learning with Textual Entailment Feedback. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2023, Toronto, Canada, July 9-14, 2023*, Anna Rogers, Jordan L. Boyd-Graber, and Naoaki Okazaki (Eds.). Association for Computational Linguistics, 6252–6272. https://doi.org/10.18653/V1/2023.ACL-LONG.344

[33] Mike Schuster and Kuldip K. Paliwal. 1997. Bidirectional recurrent neural networks. *IEEE Trans. Signal Process.* 45, 11 (1997), 2673–2681. https://doi.org/10.1109/78.650093

[34] Hinrich Schütze, Christopher D Manning, and Prabhakar Raghavan. 2008. *Introduction to information retrieval*. Vol. 39. Cambridge University Press Cambridge.

[35] StackExchange. 2023. Stack Exchange Data Explorer. https://data.stackexchange.com/.

[36] StackOverflow. 2023. Stack Overflow. https://stackoverflow.com/.

[37] Ferdian Thung, Shaowei Wang, David Lo, and Julia Lawall. 2013. Automatic recommendation of API methods from feature requests. In *2013 28th IEEE/ACM International Conference on Automated Software Engineering, ASE 2013, Silicon Valley, CA, USA, November 11-15, 2013*, Ewen Denney, Tevfik Bultan, and Andreas Zeller (Eds.). IEEE, 290–300. https://doi.org/10.1109/ASE.2013.6693088

[38] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurélien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. 2023. LLaMA: Open and

Efficient Foundation Language Models. *CoRR* abs/2302.13971 (2023). https://doi.org/10.48550/ARXIV.2302.13971 arXiv:2302.13971

[39] Aäron van den Oord, Yazhe Li, and Oriol Vinyals. 2018. Representation Learning with Contrastive Predictive Coding. *CoRR* abs/1807.03748 (2018). arXiv:1807.03748 http://arxiv.org/abs/1807.03748

[40] Hongwei Wei, Xiaohong Su, Weining Zheng, and Wenxin Tao. 2023. Documentation-Guided API Sequence Search without Worrying about the Text-API Semantic Gap. In *IEEE International Conference on Software Analysis, Evolution and Reengineering, SANER 2023, Taipa, Macao, March 21-24, 2023*, Tao Zhang, Xin Xia, and Nicole Novielli (Eds.). IEEE, 343–354. https://doi.org/10.1109/SANER56733.2023.00040

[41] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. 2022. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. In *NeurIPS*. http://papers.nips.cc/paper_files/paper/2022/hash/9d5609613524ecf4f15af0f7b31abca4-Abstract-Conference.html

[42] Moshi Wei, Nima Shiri Harzevili, Yuchao Huang, Junjie Wang, and Song Wang. 2022. CLEAR: Contrastive Learning for API Recommendation. In *44th IEEE/ACM 44th International Conference on Software Engineering, ICSE 2022, Pittsburgh, PA, USA, May 25-27, 2022*. ACM, 376–387. https://doi.org/10.1145/3510003.3510159

[43] Frank Wilcoxon. 1992. Individual comparisons by ranking methods. In *Breakthroughs in statistics: Methodology and distribution*. Springer, 196–202.

[44] Zhenyu Wu, Yaoxiang Wang, Jiacheng Ye, Zhiyong Wu, Jiangtao Feng, Jingjing Xu, and Yu Qiao. 2023. OpenICL: An Open-Source Framework for In-context Learning. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics: System Demonstrations, ACL 2023, Toronto, Canada, July 10-12, 2023*, Danushka Bollegala, Ruihong Huang, and Alan Ritter (Eds.). Association for Computational Linguistics, 489–498. https://doi.org/10.18653/V1/2023.ACL-DEMO.47

[45] Xin Xia, Lingfeng Bao, David Lo, Pavneet Singh Kochhar, Ahmed E. Hassan, and Zhenchang Xing. 2017. What do developers search for on the web? *Empir. Softw. Eng.* 22, 6 (2017), 3149–3185. https://doi.org/10.1007/s10664-017-9514-4

[46] Rensong Xie, Xianglong Kong, Lulu Wang, Ying Zhou, and Bixin Li. 2019. HiRec: API Recommendation using Hierarchical Context. In *30th IEEE International Symposium on Software Reliability Engineering, ISSRE 2019, Berlin, Germany, October 28-31, 2019*, Katinka Wolter, Ina Schieferdecker, Barbara Gallina, Michel Cukier, Roberto Natella, Naghmeh Ramezani Ivaki, and Nuno Laranjeiro (Eds.). IEEE, 369–379. https://doi.org/10.1109/ISSRE.2019.00044

[47] Wenkai Xie, Xin Peng, Mingwei Liu, Christoph Treude, Zhenchang Xing, Xiaoxin Zhang, and Wenyun Zhao. 2020. API method recommendation via explicit matching of functionality verb phrases. In *ESEC/FSE '20: 28th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Virtual Event, USA, November 8-13, 2020*, Prem Devanbu, Myra B. Cohen, and Thomas Zimmermann (Eds.). ACM, 1015–1026. https://doi.org/10.1145/3368089.3409731

[48] Jacob Yerushalmy. 1947. Statistical problems in assessing methods of medical diagnosis, with special reference to X-ray techniques. *Public Health Reports (1896-1970)* (1947), 1432–1449.

[49] Weizhao Yuan, Hoang H. Nguyen, Lingxiao Jiang, Yuting Chen, Jianjun Zhao, and Haibo Yu. 2019. API recommendation for event-driven Android application development. *Inf. Softw. Technol.* 107 (2019), 30–47. https://doi.org/10.1016/j.infsof.2018.10.010

Received 9 January 2024; revised 9 August 2024; accepted 30 August 2024