API Knowledge Graph Construction Based on Multi-Source Information Fusion

Ruilian Zhao[1], Zijie Che[1], Zhan Ma[1], Weiwei Wang[2]
[1]Beijing University of Chemical Technology, Beijing, China
[2]Beijing Institute of Petrochemical Technology, Beijing, China
Email: rlzhao@mail.buct.edu.cn, wangweiwei@bipt.edu.cn

***Abstract*—API-related knowledge is typically dispersed across various sources of information, including API documentation, Q&A forums, and other unstructured texts. This fragmentation of knowledge makes it challenging for developers to effectively query and retrieve APIs. In this paper, an API knowledge graph construction method based on multi-source information fusion is proposed to overcome these issues and enhance API retrieval. Specifically, the API-related knowledge is acquired from multiple sources, including API documentation and Stack Overflow, where API documentation describes the function and structure of APIs from designers' perspective, and Stack Overflow provides insights into the purpose and usage scenarios of APIs from users' perspective. They complement each other and together provide support for API query and retrieval. By analyzing API documentation, the corresponding APIs and domain concepts are extracted as entities and relationships between them are identified. Moreover, to extract Q&A entities from Stack Overflow, machine learning is adopted to classify the purpose of the question and performs the summary generation for its answers. Since there exists a gap between the entities from API documentation and Stack Overflow, a fusion method is raised to establish connections between them, constructing a more comprehensive API knowledge graph. To verify the effectiveness of our API knowledge graph construction method, we evaluate it in terms of the accuracy of knowledge extraction and API recommendation. The experimental results demonstrate that our API knowledge graph can significantly improve the efficiency and effectiveness of API recommendation.**

***Index Terms*—API knowledge graph, Multi-source information, Knowledge extraction, Knowledge fusion, API retrieval**

I. Introduction

API (Application Programming Interface) plays a critical role in software development. According to statistics, 87% of developers frequently leverage APIs to address diverse programming issues [1]. However, retrieving and finding suitable APIs is still a challenging task. To improve the efficiency and quality of API retrieval, researchers have built corresponding API recommendation systems from various resources to assist developers in solving programming issues related to APIs.

Currently, several typical API recommendation systems have been developed, such as RASH [2], BIKER [1], RACK [3] etc. RASH leverages lexical similarity to recommend APIs based on API documentation and the Stack Overflow (*SO*). In contrast, BIKER utilizes semantic similarity by combining API documentation and *SO* to recommend APIs. RACK establishes relationships between keywords in titles of *SO* and API to recommend APIs. These methods have enhanced retrieval efficiency in contrast to conventional API retrieval. However, they solely concentrate on valid APIs utilized in resolved problems and overlook the interconnection between APIs. In fact, different types of relationships between APIs, such as inheritance between classes and invocation between methods, may have varying impacts on API recommendation. Additionally, APIs that resolve identical problems may possess functionally similar relationships with one another, which could enhance the effectiveness of API retrieval. Nevertheless, the previous API recommendation techniques have not fully leveraged such relationships.

The Knowledge Graph is a knowledge network that can effectively represent the semantic association between information, which is suitable for expressing API-related knowledge. For example, Liu et al. [4] constructed an API knowledge graph by extracting relevant knowledge from API documentation and Wikipedia, while Li et al. [5] constructed an API warning knowledge graph by extracting warning statements from API documentation and API tutorials. Ling et al. [6] constructed an API knowledge graph based on open-source projects, which took APIs involved in projects as entities, and the calls, returns and implementations between APIs as relationships. As can be seen, API documentation can provide the dependency between APIs, Wikipedia can provide the concept of software engineering, and open-source projects can provide the relationship (call, return, implementation etc.) between APIs. However, all of them lack descriptions on actual API usage scenarios, which hinders the practical use of API knowledge graphs in solving real programming issues.

Stack Overflow is an IT technical Q&A website for programmers. It aims to help solve the actual problems of developers, and provide information about the purpose of API usage and real usage scenarios [7]. If the actual usage scenarios of APIs in *SO* are incorporated into the knowledge graph, it will greatly facilitate API retrieval for users. But *SO* suffers from a lack of clarity of purpose and information overload. The statistics show that more than 37% of *SO* questions contain more than one answer, with an average of more than 789 words per answer [8]. This makes it more difficult to capture useful knowledge from *SO*.

Thus, this paper proposes an API Knowledge Graph construction based on Multi-Source Information Fusion (AKG-MSIF), which synthesizes APIs and usage scenarios from

DOI reference number: 10.18293/DMSVIVA2024-027

API documentation and stack overflow. In particular, it entails extracting API and domain concepts as entities from API documentation and establishing relationships, such as inclusion, inheritance, and overloading, between them. More importantly, for *SO*, to extract its Q&A entities, our method uses machine learning to classify the purpose of the question and performs the summary generation for its answers. On this basis, multi-source knowledge is integrated to construct an API knowledge graph. Since there exists a gap between the entities from API documentation and *SO*, a fusion method is raised to establish connections between them. To validate the effectiveness and efficiency of our method, the constructed API knowledge graph is evaluated from two perspectives: knowledge extraction accuracy and recommendation effect. The experimental results show that compared with existing studies, our AKG-MSIF approach improve the API recommendation effectiveness and efficiency. Our contributions are as follows:

1. A novel API knowledge graph is constructed by integrating information from both API documentation and *SO*, facilitating API retrieval for users.
2. Due to unclear purpose and information overload in *SO*, a machine learning-based method is raised to classify the purpose of the question and performs the answer summary generation to obtain the Q&A entities. In addition, a knowledge fusion methods are raised to bridge the gap between entities of API documentation and *SO*.
3. To validate our approach, a series of experiments are conducted, and the experimental results show that compared with existing API recommendation systems, our novel knowledge graph has enhanced the recommendation effectiveness and efficiency.

The rest of this paper is organized as follows: Section 2 introduces the background of related techniques. Section 3 describes our method in detail. Section 4 verifies the validity of the approach. Section 5 summarizes the whole paper.

II. Related Work

Currently, several typical API recommendation systems have been developed, such as RASH [2], BIKER [1], RACK [3] etc. RASH leverages lexical similarity to recommend APIs based on API documentation and the Stack Overflow. In contrast, BIKER utilizes semantic similarity by combining API documentation and Stack Overflow to recommend APIs. RACK establishes relationships between keywords in titles of Stack Overflow and API to recommend APIs. These methods have enhanced retrieval efficiency in contrast to conventional API retrieval.

Ye et al. [9] proposed a rule-based entity extraction method, which mostly uses keywords, central words, superlatives, subordinate words, punctuation marks and other features in the text. This approach relies on the creation of a complete knowledge base and lexicon. Stephen et al. [10] proposed an approach for entity extraction through NLP and pattern-matching to classify Stack Overflow sentences. This approach also requires the design of extraction rules. Unlike the rule-based approach, it takes the syntax and grammar of the text as the focus, converts it into a syntactic dependency tree through NLP techniques and analyzes its dependencies, thereby obtaining the structural parts of the text such as noun phrases and verb phrases. For example, Lin et al. [11] manually defined 157 grammatical templates for the language style of the Stack Overflow. It can be seen that this method works better for texts with more uniform content formatting.

III. API Knowledge Graph Construction Based on Multi-Source Information Fusion

In this paper, we propose an API knowledge graph construction based on multi-source information fusion, where the API-related knowledge derives from API documentation and *SO*. The framework of our approach is shown in Fig.1, which mainly consists of knowledge acquisition and knowledge fusion. Concretely, in knowledge acquisition, APIs and corresponding domain concepts are extracted from the API documentation and taken as entities. And relationships between them, such as inclusion, inheritance, and overloading, are established. Furthermore, Q&A and API concepts are identified from *SO* by using machine learning and regarded as entities. And relationships between Q&A and API concepts are built. In knowledge fusion, multi-source knowledge from API documentation and *SO* is integrated to construct a more comprehensive API knowledge graph based on these entities and relationships. Since there exists a gap between the entities from API documentation and *SO*, the relationship between them is established by various fusion strategies. In the following, we will detail each part of our approach.

A. Knowledge Acquisition from API Documentation

API documentation provides functional descriptions and structural information (such as method, parameters and return values etc.) for APIs. This part focuses on the knowledge representation and extraction of API documentation about the PyTorch framework.

1) Knowledge representation of API documentation: The structural information of the API refers to modules, classes, methods/functions, etc., which are related to each other by inclusion, inheritance, overloading, etc. Moreover, the functional description in API documentation implies the application domains of the API, which indirectly reflect the relationship between the API and application domain. Both the functional description and structural information can provide useful guidance in API retrieval. Thus, we use the domain concept and API modules, classes, methods/functions to express the API knowledge in the document. Further, the APIs and domain concepts can be associated through ”refer to”, which means that the description of an API mentions the corresponding domain concept. So, this paper regards the API and domain concepts as entities in the API documentation and their ”refer to” as the relationship between them.

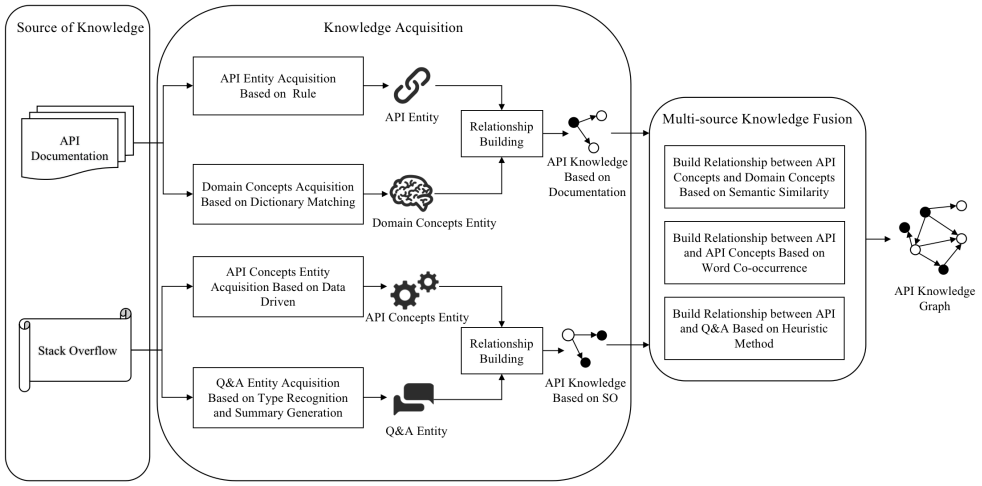

Fig. 1. The Framework of API Knowledge Graph Based on Multi-Source Information Fusion

2) Knowledge extraction from API documentation: To extract API entities, the API documentation is analyzed. And we found that API documentation is semi-structured data, where different HTML tags represent different types of API entities, such as functional descriptions, parameters, return values, and return value types etc. Thus, API entities are recognized through HTML tags. Further, the relationships between API entities include inclusion, inheritance, overloading. According to the declaration rules of the class, regular expressions are used to extract the inheritance relationship, and syntax analysis is employed to extract the inclusion and overloading relationship.

Furthermore, each API corresponds to a functional description, and the functional description implies domain concepts. Thus, for the application domain in API function descriptions, we use existing domain concept dictionary [12] and NLP to match and recognize them. And the "refer to" relationship between the API and the domain concept can be extracted from this corresponding structure. For example, the functional description of API "*torch.normal*" contains the domain concept "*standard deviation*". When the domain concept "s*tandard deviation*" is identified, a "refer to" relationship can be established between the API "*torch.normal*" and the domain concept "*standard deviation*".

B. Knowledge Acquisition from Stack Overflow

Stack Overflow provides many information (such as title, the body of the question, label, accepted answer etc.) which contains specific application scenario information of the API. This part mainly focuses on the knowledge representation and extraction on the Q&A tagged by "PyTorch" on the Stack Overflow.

1) Knowledge Representation of Stack Overflow: The Q&A information of *SO* contain terms related to the API, where these terms are related to software development, without being limited to a specific field. Intuitively, these terms implicitly abstract and summarize the functional role of a specific API. Thus, these terms are adopted to express the API-related knowledge and regarded as API concept entities.

What's more important, the Q&A in *SO* describe the actual problems encountered by developers and provide answers about its usage scenario as well as the purpose of API. So the Q&A is critical in API retrieval. But the Q&A in *SO* suffers from problems of unclear purposes and information overload. The unclear purpose refers to the difficulty for Q&A to grasp the reason behind a user's question and find the corresponding answer. And information overload refers to the fact that a single question may have multiple long answers. Statistics show that over 37% of Q&As include more than one answer, and each answer has an average of over 789 words [8], making it challenging to obtain critical information from the Q&A.

Thus, in this paper, we propose a questions' purpose identification method through classifying the Q&A automatically. Further, the answers are summarised based on the features of Q&A to alleviate the issue of information overload. And the simplified Q&As that consists of the purpose of questions and summary of answers are referred to as Q&A entities. Besides, the "refer to" relationship between Q&A entities and API concept entities can be established.

2) Knowledge extraction from Stack Overflow: The API in *SO* is usually labeled with *<code>* tags. Thus, API can be recognized through matching the element labeled with this tag with the API name in the API documentation. For the concepts of API in *SO*, they often appear in the same sentence, paragraph, or Q&A with the API. Since the API concept may be a multi-word concept, such as "convolutional layer", this paper proposes a frequency-based API concept recognition method. Concretely, we look for words that often appear consecutively but not often separately in *SO* through NLP, and take them as API concepts. NLP is used to segment and remove stop words from *SO* Q&A to form single-word concepts. According to the frequency of consecutive words, we calculate the phrase score and extract the API concept.

The phrase score is shown in formula (1):

$$score(w_i w_j) = \frac{count(w_i w_j) - \delta}{count(w_i) \times count(w_j)} \quad (1)$$

Where $count(w_i w_j)$ represent the number of times two consecutive words w_i and w_j appear in the whole documentation. $count(w_i)$ and $count(w_j)$ represent the number of times the words w_i and w_j appear. δ is a threshold. When the frequency of the two consecutive words w_i and w_j is less than δ, w_i and w_j cannot form a two-word phrase. When a two-word concept is formed, formula (1) can be repeated to detect three-word phrases. Since API concepts consisting of more than three words are uncommon, this paper only recognizes phrases of up to three words as API concepts.

To extract the Q&A entities from *SO*, this paper employs machine learning to classify the purpose of the questions and obtain the purpose type. Based on this, the answer summary generation based on feature extraction is performed.

In more detail, based on the categorization of *SO* Q&A by Stefanie Beyer et al.'s [13], this paper divides the Q&A into seven categories based on the purpose of question as follows: (1) "API USAGE" class is to seek suggestions for implementing a feature or API; (2) "DISCREPANCY" class is to request Code segments to resolve unexpected results; (3) "ERRORS" class is to request a bug fix or handle an exception; (4) "REVIEW" class is to request the best solution; (5) "CONCEPTUAL" class is to ask about the rationale or background of the API; (6) "API CHANGE" class is to seek solutions to issues arising from API version changes problems; (7) "LEARNING" class is to ask for documentation or tutorials to learn a tool or language.

XGBoost(eXtreme Gradient Boosting) is one of machine learning algorithms, which have the capability of fast learning and prediction [14]. Therefore, in this paper, XGBoost algorithm is used to train classifiers for *SO* questions to determine the purpose of questions. The main steps include: (1) Label *SO* Q&A into one of the seven categories. (2) Convert questions into corresponding word lists through NLP including as segmentation, stop word removal, and lemmatization. (3) The TF-IDF reflects the importance of a word by its frequency, where *TF* (term frequency) measures the frequency that a term appears in a document and *IDF* (the inverse document frequency) estimates the ratio of total documents to the documents that contain the term. In this paper, the TF-IDF of a question is used as its textual feature and fed into the XGBoost algorithm to identify the type of the question.

Furthermore, to address information overload in answers of *SO*, this paper generates summaries for answers based on the relevant paragraphs in the answers. That is, based on the characteristics of *SO*, the relevance of each paragraph to the question is calculated by combining question-related features, content-related features, and user features, and the top M paragraphs are selected as the summary of the answer. The concrete feature analysis is as follows:

(1) Question-related feature: if a paragraph contains key words from the question, it is considered to be related to the question. The more key words a paragraph contains, the higher its relevance. In this paper, tags of *SO* are used as the set of key words. And the relevance of each answer paragraph and the question is calculated based on the ratio of the key words involved in them.

(2) Content-related feature: This feature evaluates the importance of content of paragraphs from three sub-features: the API occurrence, information entropy, and semantic templates. For the API occurrence, if at least one API appears in the paragraph, this sub-feature value is set to 1. Otherwise, it is set to 0. For information entropy, the inverse documentation frequency (IDF) value of a word can be used to measure its information entropy, which can be calculated using formula (2), where p represents the total number of paragraphs and p' represents the number of paragraphs containing a particular word. The higher the IDF value, the lower the occurrence frequency of the particular word, indicating greater importance. The entropy value of a paragraph can be represented by the sum of its words' IDF values, normalized to (0,1].

If a paragraph conforms to at least one semantic template, the sub-feature value is set to 1. Otherwise, it is set to 0. The feature value of the content is the sum of the three sub-feature values.

$$IDF = log(\frac{p}{p' + 1}) \quad (2)$$

(3) User feature: In *SO*, each answer has a corresponding vote, and the higher the vote, the higher the quality of the answer. Therefore, the number of votes for the current answer indicates the importance of the paragraph in this answer, which can be regarded as the user feature.

For the above three features, we add a smoothing factor of 0.0001 to avoid the feature score of 0. All features are normalized to (0,1], and the normalized values of each feature are multiplied together to obtain the total score of each paragraph. Finally, the top M paragraphs are selected as the summary of the answer.

By identifying the type of question and generating the answer summary, an valid Q&A entity can be obtained. Furthermore, since a Q&A usually mentions multiple API concepts, a "refer to" relationship can be also established between the API concept and the Q&A entities.

C. Knowledge Fusion from API Documentation and Stack Overflow

To construct a complete API knowledge graph, the API knowledge from API documentation and *SO* Q&A website should be integrated. As there is a gap between entities from the API documentation and *SO*, a fusion method is proposed to establish a link between them. As mentioned above, entities about APIs and corresponding domain concepts are extracted from the API documentation. And entities about API concepts and Q&A are extracted from *SO*. Since domain concepts are not directly related to Q&A entities, it is not mandatory to establish a connection between them. Thus, this paper performs knowledge fusion between entities about API and

API concept, API concept and domain concept, and API and Q&A.

1) Fusion between entities of API and API concepts based on word co-occurrence: Intuitively, API concepts abstract and summarize the functional role of a specific API. Thus, semantic relationships exist between them. In fact, API and API concepts usually co-occurs in the same paragraph, so word co-occurrence can be used to link them. Co-occurrence frequency can evaluate the degree of correlation between API and API concepts, which refers to the number of times the API and API concepts appear in the same paragraph. Therefore, this paper captures the semantic relationship between API and API concepts by calculating their co-occurrence frequency. Its formula is shown in formula (3), where $freq(A_i \rightarrow Ac_j)$ represents the co-occurrence frequency between API A_i and API concept AC_j, and α is the threshold. If the co-occurrence frequency is not lower than the threshold α, a "refer to" relationship can be established between API A_i and API concept AC_j.

$$freq(A_i \rightarrow AC_j) \geq \alpha \tag{3}$$

2) Fusion between entities of API concept and domain concept based on semantic similarity: The relationship between API concepts and domain concepts can help establish indirect connections between the API, which can help improve the possibility of retrieving relevant APIs. Thus, it is necessary to build the links between them. Since API concepts and domain concepts are composed of phrases, their relationship can be determined by combining lexical and semantic similarity. When the similarity between them is higher than the given threshold, their "related to" relationship can be established.

In more detail, the lexical similarity sim_{lex} can be calculated using Jaccard similarity, as shown in formula (4), where $Token(n)$ represents the words that make up the concept. The semantic similarity between n_1 and n_2 is calculated using formula (5), where $V_p(n_1)$ represents the vector of the concept entity, and sim_{cos} represents the cosine similarity between the two vectors. In this paper, based on SO Q&A and API documentation corpora, we use word2vec [15] to train a word embedding model and convert concepts into word vectors. Based on the lexical and semantic similarity, a weighted similarity calculation formula is raised, which is shown in formula (6). Generally, semantic similarity is more important than lexical similarity, so $w_1 < w_2$ is set.

$$sim_{lex}(n_1, n_2) = \frac{|Token(n_1) \bigcap Token(n_2)|}{|Token(n_1) \bigcup Token(n_2)|} \tag{4}$$

$$sim_{con}(n_1, n_2) = \frac{sim_{cos}(V_p(n_1), V_p(n_2)) + 1}{2} \tag{5}$$

$$sim(n_1, n_2) = w_1 \times sim_{lex}(n_1, n_2) + w_2 \times sim_{con}(n_1, n_2) \tag{6}$$

Formula (4) measures the similarity between domain concepts and API concepts in terms of both lexical and semantic aspects, where n_1 and n_2 represent the candidate domain concept and API concept, respectively.

3) Fusion between entities about API and Q&A based on heuristic algorithm: The relationship between API entity and Q&A entity enables the integration of the general knowledge of the API (such as functional description, parameters, return values, etc.) with their specific knowledge in concrete usage scenarios (such as how to solve specific problems), which provides developers with a more comprehensive API information. Thus, a fusion strategy is raised to establish the relationship between them.

However, APIs mentioned in *SO* Q&A are not always in the form of fully qualified names. For example, the API "forward()" is mentioned in *SO* in answer to the question "what does model.train() do in PyTorch". But "forward()" can be associated with multiple APIs. In order to establish an unambiguous correlation between Q&A entities and corresponding API entities, we design a heuristic strategy. In general, in *SO* Q&A, the appearance of code elements has locality, i.e., APIs mentioned in the same Q&A usually belong to the same module or class. By parsing the tag in HTML, we can identify the module or class of the API mentioned in the Q&A. By specifying regular expressions to identify the module or class in the code block, their APIs can be determined. Once unambiguous APIs are identified, a "refer to" relationship can be established between the Q&A entity and the API entity.

IV. Experimental Analysis

In order to verify the validity of our AKG-MSIF approach for API retrieval, we conduct a series of experiments on PyTorch API documentation and 7043 API Q&A on Stack Overflow, and the effectiveness and efficiency are evaluated on the basis of these experiments. To assess our approach, three research questions are raised as below.

- **RQ1.** Can the API knowledge be accurately extracted from multi-source information?
- **RQ2.** Can our integrated API knowledge graph obtained by fusing API documentation and *SO* improve the effectiveness of API retrievals?
- **RQ3.** How effective is our AKG-MSIF approach in the API recommendation? How much improvement can be achieved compared to baseline methods?

A. Experimental Subject

In this paper, we extracted questions and answers marked as "PyTorch" from the official *SO* data (data released as of June 2022). We extracted 7043 questions and answers labeled as "PyTorch" as the subjects. In order to ensure the quality of the Q&A, we excluded the Q&A with no answer and those with a rating of less than 1 (indicating that the content of the answer was not accepted), and finally collected 3361 Q&A with good quality. Besides, we develop a crawler script based on scrapy framework, and obtain information about the API by crawling the official API documentation of PyTorch. In total, 27 modules, 314 classes, 1570 functions or methods and their corresponding basic description information are extracted. The fully qualified names of these API classes

and functions/methods are used to build the API dictionary of PyTorch.

The final constructed API knowledge graph includes 28730 entities and 142,578 relations. Among them, there are 1912 API entities, 16216 API concepts, 7116 domain concepts, 3361 Q&A entities.

B. Experimental Design

When extracting API concepts, the threshold δ of the frequency of the two consecutive words was set to 5 to avoid the recognition of uncommon phrases. Furthermore, when integrating API and API concepts, the co-occurrence frequency threshold is set to 3 to capture the semantic association between entities about API and API concept. And when integrating API concepts and domain concepts, considering that semantic similarity is more important than lexical similarity, weights w_1 and w_2 were set to 0.3 and 0.5 respectively.

Besides, in order to create experimental queries for retrieving knowledge graph, the following selection criteria were used: 1) The questions had a rating at least 1. 2) The answer to the question contains the explicit and exact API and the title of the question does not contain the API. Based on them, 10 questions were randomly selected from the PyTorch-related questions in *SO* as the queries for the experiment, and the corpus for constructing the knowledge graph did not contain these 10 questions in order to ensure that the search of API knowledge graph was valid.

The API knowledge graph constructed in this paper is stored in a Neo4j graph database. For queries, corresponding keywords are extracted by syntactic analysis using the StanfordCoreNLP[21]. And for each keyword, we search a semantically similar concept entity in the API knowledge graph. The API entity with a "refer to" relationship with the concept entity is used as a candidate API, and the Q&A associated with the candidate API is information about the specific usage scenario. Since there may be multiple candidate APIs, they are ranked according to their semantic relevance to the query. That is, the APIs are ranked by calculating the semantic similarity (formula(5)) between the query and each candidate API function description, and the top K APIs are recommended to users.

The related APIs obtained by searching the API knowledge graph are further analyzed, so as to verify the effectiveness of API recommendation based on our knowledge graph. In particular, we invite 10 masters from the same lab with two years of experience in using PyTorch to analyze the accepted or highly rated responses to these questions together with the authors themselves. When disagreements arose, consistent conclusions were drawn by analyzing the official API documentation. The final 10 experimental queries and the number of correct APIs for them are shown in Table 2.

C. Experimental Results and Analysis

1) Results for RQ1: The focus of this experiment is to demonstrate the effectiveness of knowledge extraction from the perspective of entities and relationships extraction. Thus, the accuracy of entities and relationships extracted from the API documentation and *SO* is evaluated.

As is known, API entities and their relationships are derived from semi-structured API documentation. Based on specific HTML tags and declarations, entities and relationships related to them can be extracted and validated easily. Therefore, this paper mainly evaluates the extraction accuracy of entities from unstructured text, namely API concept, domain concept and Q&A entities, as well as their relationships. Since the number of domain concept entities and relations exceeds tens of thousands, it takes a lot of time to check all entities and relations. Thus, this paper adopts the random sampling. In more detail, random samples of 5% of the entities or relationships from the constructed API knowledge graph is selected with a 95% confidence level, and the sample estimation accuracy has an error margin of 0.05.

To assess the validity of API concepts and domain concepts, we manually identifying the accuracy of sampling results. After random sampling, the accuracy of 356 domain concepts obtained from domain concept entities is up to 95.6%, and the error mainly comes from the domain concept dictionary itself. The accuracy of 800 API concepts sampled from API concept entities is 97.8%, and the main reasons affecting the accuracy are some numerical indicators often mentioned in *SO* Q&A, such as "200k images". These terms should not be identified as API concepts. To evaluate the validity of the relationship between the API concept and domain concept, the accuracy of sampling results is also manually identified. After random sampling, 4000 relationships were obtained from the API knowledge graph, of which 94.3% of API concepts and domain concept semantics were identified as relevant. The missing relationships are due to API concepts or domain concepts not being correctly identified.

To evaluate the effectiveness of the Q&A entity extraction, the accuracy of the classification of *SO* questions and the quality of answer summary is measured. For the classification of *SO* question, the XGBoost algorithm is used to classify the question types. Through manual labeling, 326 labeled Q&A were obtained, including 118 "API USAGE", accounting for 36.2%, and 65 "CONCEPTUAL", accounting for 20%; 45 "DISCREPANCY", accounting for 13.9%; 34 "ERRORS", accounting for 10.4%; 24 "REVIEW", accounting for 6.1%. The number of "API CHANGE" and "LEARNING" is 20, accounting for 6.1%. In this paper, a 10-fold cross-validation method was used to verify the validity of *SO* Q&A classification. The classification effectiveness was evaluated using precision, recall, F1 value and accuracy. To verify the advantages of XGBoost-based classification, the experiment uses SVM (Support Vector Machine) and RF (Random Forest) as comparison methods. The comparison of classification effectiveness of different algorithms are shown in Table II. The precision of XGBoost is improved by 14.6% and 5.7% compared to SVM and RF, respectively, and the accuracy is improved by 5.9% and 4.6%, respectively. Therefore, it can be seen that the classification for questions of *SO* using XGBoost

TABLE I
QUERIES AND THE NUMBER OF STANDARD ANSWERS

SO Number	Question	Number of Related API
44524901	How to do product of matrices in PyTorch?	6
54716377	How to do gradient clipping in PyTorch?	2
48152674	How to check if PyTorch is using the GPU?	7
50544730	How do I split a custom dataset into training and test datasets	1
55546873	How do I flatten a tensor in PyTorch?	4
53841509	How does adaptive pooling in PyTorch work?	4
53266350	How to tell PyTorch to not use the GPU?	2
53879727	PyTorch-How to deactivate dropout in evaluation mode?	2
51136581	How to do fully connected batch norm in PyTorch?	4

algorithm is better than other methods.

TABLE II
COMPARISON OF CLASSIFICATION EFFECTIVENESS OF DIFFERENT ALGORITHMS

Method	Precision	Recall	F1 Score	Accuracy
XGBoost	0.871	0.847	0.834	0.910
SVM	0.760	0.851	0.785	0.850
RF	0.824	0.903	0.849	0.870

Furthermore, to evaluate the quality of summary generation, this part measures the quality of summary in terms of relevance, usefulness, and diversity. Relevance indicates whether the summary is relevant to the question. Usefulness indicates whether the summary content can solve the problem, and diversity indicates whether the question can be answered from multiple perspectives. In this part, we set a maximum score of 5 and a minimum score of 1 for each indicator. To evaluate the quality of the summary, master students with two years of experience using PyTorch were invited to participate in the evaluation. Table III represents the evaluators' assessment of the 10 *SO* Q&A summaries in Table I, where the three evaluation metrics for each query ranged from 3 to 5, and the average results for the 10 questions were 3.6, 3.4, and 3.7, respectively, indicating that the feature extraction-based summary generation method is effective.

2) Results for RQ2: To validate whether the integrated API knowledge graph obtained by fusing API documentation and *SO* improve the effectiveness of API retrievals, we compare the retrieval results based on multi-source API knowledge graph with single-source API knowledge graph. The single-source API knowledge graph is constructed by extracting API-related knowledge from API documentation and *SO* Q&A websites, respectively. The single-source API knowledge graph extracted from the API documentation includes API entities, domain concept entities and the relationships among them; And the other single-source API knowledge graph extracted from *SO* Q&A website includes API entities, API concept entities, Q&A entities and the relationships among them. Three commonly used metrics in information retrieval are selected to evaluate the effectiveness of API retrieval, namely, HR(Hit Ratio), MRR(Mean reciprocal rank) and MAP(Mean average precision). *HR* evaluates the percentage of correct results out of all correct results in the top K search results. *MRR* is the position where the first correct result appears. *MAP* is the ranking of all correct results. Since the number of the API related to query is less than 10, this paper sets K=10.

The experimental results are shown in Table IV. It can be seen that the API recommendation effectiveness of multi-source API knowledge graph is better than that of single-source API knowledge graph. This is because that information fusion indirectly associates the API related to the questions through the mentioned concepts, which improves the recommendation effectiveness. In addition, it is worth noting that there is a large difference between the recommendation results based on API documentation and those based on *SO* Q&A websites. The reason is that the functional descriptions provided by the API documentation do not involve specific usage scenarios. Thus, it is difficult to match the domain concepts with the keywords in the specific questions, resulting in unsatisfactory recommendation results by using only the API documentation. In a summary, we can see that our API knowledge graph is more comprehensive, and enhances the effectiveness of API retrieval, indicating our AKG-MSIF approach is effective.

3) Results for RQ3: To evaluate the effectiveness of our AKG-MSIF approach in API recommendation, three metrics including HR, MRR and MAP are also used. Besides, the BIKER recommendation system also combines two types of data sources (API documentation and Stack Overflow), and it uses the same dataset as the approach in our paper. While RACK recommendation system uses only Stack Overflow data sources. These two techniques are used as our comparison methods. The experimental results are shown in Table V. It can be seen that the HR index of our AKG-MSIF has increased by 49% compared with BIKER and 87% compared with RACK. The MRR index has increased by 22% compared with BIKER and 52% compared with RACK. This indicates that in the first 10 search results, AKG-MSIF can search more APIs related to the query, and can find the first correct API earlier than BIKER and RACK. Thus, our AKG-MSIF has improved retrieval efficiency compared with BIKER and RACK. Besides, Table VI shows the comparison in terms of time cost. The construction time of AKG-MSIF is mainly concentrated in the training of the classifier and summary generation. Although the time cost of the construction of the method in this paper is higher than that of BIKER and RACK, there is a significant improvement in the query speed and the recommendation effectiveness of the API. Thus, our AKG-MSIF approach is more effective in API retrieval.

TABLE III
Q&A SUMMARY SCORE

Question	Relevance	Usefulness	Variety
How to do product of matrices in PyTorch?	3	3	3
How to do gradient clipping in PyTorch?	2.85	3	3.57
How to check if PyTorch is using the GPU?	4	3.5	3.5
How do I split a custom dataset into training and test datasets	3.5	4	3.5
How do I flatten a tensor in PyTorch?	3.57	2.85	3.57
How does adaptive pooling in PyTorch work?	3.5	3	4
How to tell PyTorch to not use the GPU?	4.5	4	4.5
PyTorch-How to deactivate dropout in evaluation mode?	3	3	3.5
How to do fully connected batch norm in PyTorch?	4	3.5	4
How to create a normal distribution in PyTorch?	4.5	4	4
AVERAGE	**3.6**	**3.4**	**3.7**

TABLE IV
COMPARISON OF RECOMMENDATION RESULTS BETWEEN MULTI-SOURCE INFORMATION FUSION AND SINGLE-SOURCE INFORMATION

Method	HR	MAP	MRR
Only SO	0.726	0.490	0.583
Only API Doc	0.322	0.181	0.149
Both	0.774	0.558	0.701

TABLE V
COMPARISON OF RECOMMENDATION EFFECTIVENESS BY DIFFERENT METHODS

Method	HR	MAP	MRR
AKG-MSIF	0.774	0.558	0.701
BIKER	0.520	0.521	0.573
RACK	0.415	0.420	0.462

TABLE VI
COMPARISON OF TIME COST BY DIFFERENT METHODS

Method	Cost	Query Cost
AKG-MSIF	16 min	1s/query
BIKER	5 min	2s/query
RACK	10min	5s/query

V. CONCLUSION

This paper proposes an API knowledge graph construction approach based on multi-source information fusion(AKG-MSIF), which integrates the functional and structural information of APIs, as well as the specific usage scenarios of APIs from documentation and *SO* Q&A websites. The experiment validated the effectiveness of our approach from two aspects: information extraction and API recommendation effectiveness. And the results show that the accuracy of domain concept identification is up to 95.6%, and that of API concepts is 97.8%. And 94.3% of relationships between API concepts and domain concept are correctly identified. Meanwhile, the Q&A entities from *SO* are identified effectively by machine learning and summary generation. Furthermore, compared with existing API recommendation systems, our API knowledge graph is more comprehensive, enhancing the effectiveness of API retrieval.

REFERENCES

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

[2] J. Zhang, H. Jiang, Z. Ren, and X. Chen, "Recommending apis for api related questions in stack overflow," *IEEE Access*, vol. 6, pp. 6205–6219, 2017.

[3] M. M. Rahman, C. K. Roy, and D. Lo, "Rack: Automatic api recommendation using crowdsourced knowledge," in *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, vol. 1. IEEE, 2016, pp. 349–359.

[4] M. Liu, X. Peng, A. Marcus, Z. Xing, W. Xie, S. Xing, and Y. Liu, "Generating query-specific class api summaries," in *Proceedings of the 2019 27th ACM joint meeting on European software engineering conference and symposium on the foundations of software engineering*, 2019, pp. 120–130.

[5] H. Li, S. Li, J. Sun, Z. Xing, X. Peng, M. Liu, and X. Zhao, "Improving api caveats accessibility by mining api caveats knowledge graph," in *2018 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 2018, pp. 183–193.

[6] C.-Y. Ling, Y.-Z. Zou, Z.-Q. Lin, and B. Xie, "Graph embedding based api graph search and recommendation," *Journal of Computer Science and Technology*, vol. 34, pp. 993–1006, 2019.

[7] C. Treude and M. P. Robillard, "Augmenting api documentation with insights from stack overflow," in *Proceedings of the 38th International Conference on Software Engineering*, 2016, pp. 392–403.

[8] S. Nadi and C. Treude, "Essential sentences for navigating stack overflow answers," in *2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 2020, pp. 229–239.

[9] D. Ye, Z. Xing, J. Li, and N. Kapre, "Software-specific part-of-speech tagging: An experimental study on stack overflow," in *Proceedings of the 31st Annual ACM Symposium on Applied Computing*, 2016, pp. 1378–1385.

[10] S. Soderland, "Learning information extraction rules for semi-structured and free text," *Machine learning*, vol. 34, pp. 233–272, 1999.

[11] B. Lin, F. Zampetti, G. Bavota, M. Di Penta, and M. Lanza, "Pattern-based mining of opinions in q&a websites," in *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. IEEE, 2019, pp. 548–559.

[12] C. Wang, X. Peng, M. Liu, Z. Xing, X. Bai, B. Xie, and T. Wang, "A learning-based approach for automatic construction of domain glossary from source code and documentation," in *Proceedings of the 2019 27th ACM joint meeting on european software engineering conference and symposium on the foundations of software engineering*, 2019, pp. 97–108.

[13] S. Beyer, C. Macho, M. Pinzger, and M. Di Penta, "Automatically classifying posts into question categories on stack overflow," in *Proceedings of the 26th Conference on Program Comprehension*, 2018, pp. 211–221.

[14] S.-E. Ryu, D.-H. Shin, and K. Chung, "Prediction model of dementia risk based on xgboost using derived variable extraction and hyper parameter optimization," *IEEE Access*, vol. 8, pp. 177 708–177 720, 2020.

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