The Journal of Supercomputing
https://doi.org/10.1007/s11227-022-04603-3

KG2Lib: knowledge-graph-based convolutional network for third-party library recommendation

Jing-zhuan Zhao[1] · **Xuan Zhang**[1,2,3] · **Chen Gao**[4] · **Zhu-dong Li**[1] · **Bao-lei Wang**[1]

Accepted: 12 May 2022

Abstract

In the process of software system evolution, software users constantly put forward a large number of expectations. For these expectations, software developers usually use the existing third-party libraries and other software resources to accelerate their development processes. At present, tons of third-party libraries are available. Therefore, appropriate recommendation methods are very important for developers to find suitable libraries for their development projects. In this paper, we present KG2Lib, a recommendation method to assist software developers in selecting suitable software libraries for their current projects. KG2Lib exploits a knowledge-graph-based convolutional network to recommend software libraries by relying on a set of libraries which were already called by current projects. The interaction matrix, weight matrix and knowledge graph are the inputs of KG2Lib. What's more, KG2Lib recommends libraries to developers from project level and library level, which can better capture the fine-grained information to achieve better recommend performance. The performance of KG2Lib was evaluated on three datasets with four existing baseline models. The experimental results show that KG2Lib achieves better performance and helps software developers accurately select the appropriate third-party libraries.

Keywords Software development · Library recommendation · Knowledge graph · Graph convolutional network

✉ Xuan Zhang
zhxuan@ynu.edu.cn

1 School of Software, Yunnan University, Kunming 650000, China

2 Key Laboratory of Software Engineering of Yunnan Province, Kunming 650000, China

3 Engineering Research Center of Cyberspace, Kunming 650000, China

4 School of Information Science & Engineering, Yunnan University, Kunming 650000, China

Published online: 30 June 2022

1 Introduction

With the rapid development of Internet technologies, the iteration cycle of software becomes shorter [1], software developers need to develop higher quality products in a shorter cycle. Software reuse is a solution to avoid the repeated work in software development [2], which plays an important role in the process of software development. Third-party libraries are important reusable software resources in software development activities [3]. The reuse of third-party libraries can reduce the code workload of developers, enable developers to focus on the crucial parts of the project, improve software development efficiency and shorten development cycle [4]. Therefore, software developers often search the required third-party software libraries based on keyword matching on open source and various software development websites. However, the content of the third-party libraries on various blogs or forums on the web page may be incomplete, and developers need to view many different web pages to select the appropriate libraries, which greatly consumes developers' time.

One possible solution is personalized recommendation system [5]. By obtaining the relationship between users and items, it provides users with preferred items (Wang et al., 2021) [6]. In recent years, recommendation system has been applied to many fields such as music, film and e-commerce. The current recommendation technologies mainly include recommendation based on collaborative filtering (Han et al. 2021) [7], recommendation based on knowledge graph [8, 9, 10], recommendation based on deep learning [11, 12, 13], etc. By adding recommendation technology, users can reduce their time consumption in selecting their enthusiastic items [14].

Most of the existing third-party library recommendation technologies for software development are mainly based on the library use pattern (Katsuragawa et al. 2018) [8, 15, 16], or based on the library text description and developer requirements description [15, 17, 18], Deshpande et al. (2022) [19], or the relationship between the projects and the libraries [4, 20, 21]. Such recommendation methods have the following problems:

1. The recommendation results do not cover the whole set of recommended libraries. Some popular libraries are recommended more frequently [4], while others are not so popular and have the lower chance to be recommended. This is the "long tail problem" in the field of third-party library recommendation (Anderson et al., 2006) [22] and leads to limited recommendation results and lack of diversity.
2. Only considering the correlation among projects and ignoring the rich auxiliary information of the third-party libraries, the recommendation results are not refined enough.
3. Focusing on projects level (projects' similarity) or library level (library usage pattern) for the recommendation, which separated the relation between projects and libraries.

To solve the above problems, KG2Lib, a novel approach utilizing a convolutional network based on knowledge graph [23] is proposed. It exploits the rich semantic relationship of knowledge graph [24] to make the recommendation results more diversified and fine-grained. Next, the vector representation of projects is obtained from the project level, and the similarity among projects are calculated. According to the similarity value, the top-N projects and their corresponding third-party libraries are obtained. Then, the fine-grained information about the libraries from the library level are taken into consideration, which can not only consider the correlation information among projects, but also the rich auxiliary information among libraries. What's more, the improved graph convolution network (GCN) is employed to consider whether the third-party libraries in the library set can be recommended. This ensures that the libraries we recommend are highly relevant to the project, not the most popular libraries. Therefore, the "long tail problems" in third-party recommendation field can be alleviated greatly.

KG2Lib aims at providing software developers who have already called some libraries in the current project, and expect to get recommendations on which additional third-party libraries should be further called. In other words, a software developer provides the current libraries he has called, and KG2Lib returns a library list for him to choose from in the future. This scenario is done based on training data from other projects. To this end, this paper makes the following contributions:

1. A new software libraries recommendation model is introduced, which has made full use of all libraries information and projects' similarities to assist developers for choosing suitable software libraries for their projects.
2. Constructing a representation model to describe the relationship and importance among the third-party libraries and the projects. Meanwhile, the new representation model is used to compute similarities of projects.
3. Alleviate a phenomenon in recommender system that popular items are recommended frequently while less popular items are often recommended seldom or not recommended.
4. Perform an experimental study on the performance of KG2Lib in comparison with four current baseline models on three datasets, and exploiting various quality metrics, i.e., success rate, diversity and so on. The experimental results show that KG2Lib achieves better performance.

In the following, Sect. 2 introduces the background knowledge and related work of third-party library recommendation in software development. Section 3 describes the recommendation method of KG2Lib. Section 4 introduces the implementation details of model. Section 5 verifies the performance of KG2Lib through experimental analysis. Section 6 concludes the paper and outlines directions for the future work.

2 Related work

Recommender systems aims at helping users find useful and preferred information. Early recommender systems used collaborative filtering methods. These methods collected users' preferences by user–item historical interactive data, and calculated the similarities between users and items. Wang and Nie [25] generated item's recommendation by incorporating all auxiliary information about users and items, which has improved the problem of data sparsity ad scalability in previous collaborative filtering algorithms. Lei et al. (2010) [26] developed a refined item-based collaborative filtering method using the average rating for items and took users' general opinion on items into consideration. Based on Lei et al. (2010) [8, 26] work, Liu et al. (2015) [27] proposed an improved collaborative filtering recommendation algorithm based on user ratings and item attributes, and calculated their similarity to recommend items. Guan [28] adopted a weighted feature form and a Bayesian form to enhance the performance in the process of collaborative filtering to alleviate the data sparsity problem.

Most of the above systems heavily rely on historical data and have the cold start problem. On the basis of tons of data, constructing the interactive matrix, calculating the similarity about users and items, many useful and abundant auxiliary information are ignored, leading to a poor performance. After the emergence of knowledge graph [24], lots of researchers have begun to combine collaborative filtering with knowledge graph to obtain diverse and rich results [30, 31]. Zhang et al. [32] proposed a collaborative filtering with the implicit feedback based on knowledge graph, in which the interactions between users and items are modeled as an interaction knowledge graph, and the collaborative filtering problem is converted into link prediction problem. Yu et al. [33] put forward a privacy-preserving multi-task framework for knowledge graph enhanced recommendation. Dang et al. [34] proposed a service recommendation model based on knowledge graph and knowledge representation learning to alleviate the information overload and data sparsity problem in Web services.

In their model, knowledge graph can effectively enhance the recommendation results as an auxiliary tool [19]. With the development of deep learning, graph convolutional network has attracted researchers' concern [31]. The combination of knowledge graph and GCN is a direction to improve the performance of graph structure recommendation. Mei et al. [36] proposed a collaborative filtering model based on light GCN to mine various user–item interaction information. The user–item interaction graph was constructed according to the interaction history and item knowledge graph information, and then inputted it into the model. Yang et al. (2021) [35] proposed a hierarchical attention GCN, combined with the explanatory recommendation of knowledge graph to mine users' potential preferences from the high-order connected structure of heterogeneous knowledge graph. Zhang et al. (2019) [37] utilized deep convolutional neural network model to learn the attribute information of entities in knowledge graph, encode attribute information of entities, and both attribute information and triple structure information were utilized to learn knowledge representation, and then generated attribute-based representation of entities.

So far, the recommendation system has been applied in various fields. In third -party library recommendation field, Thung et al. [4] combined association rule mining with collaborative filtering technology, calculated the similarity between third-party libraries, and generated a recommendation list to developers. This is also the earliest search on third-party library recommendation. Saied et al. [38] proposed a recommender system that recommend libraries based on the extent to which they are used together. Specifically, LibCUP uses a clustering approach based on the DBSCAN algorithm to identify and recommend library co-usage patterns. Nguyen et al. (2020) [39] proposed CrossRec, which adopted a collaborative filtering technique to recommend libraries for software developers. The relation between projects and libraries are expressed as an interaction matrix, and the similarity between projects are calculated by the number of libraries they have called. Chen et al.(2020) [40] integrate knowledge graph into the third party library recommendation for mobile application development.

The above methods for third-party library recommendation did not exploit the information about both projects and libraries. The work of Thung et al. [4] adopted a filtering technique, which can only recommend popular libraries and take no consideration on the libraries. This leading a phenomenon that other unpopular libraries but have high correlation with current project are not recommended. Saied et al. [38] only considered the library usage pattern but ignored the project's actual situation. As a result, the libraries it recommends may be not meet the software developers' requirement. The collaborative filtering method in Nguyen et al. (2020) [39] only considered the similarity between projects but ignored the rich information about libraries. Our work differs from the above methods since we not only consider the projects' similarity, but also make full use of the rich libraries' information. First, the projects' similarity is calculated. Then, the interaction matrix, the library weight matrix and the library knowledge graph are inputted into the improved GCN.

3 Methodology

In this section, we describe KG2lib, a third-party library recommendation for software developers. By introducing the knowledge graph for the third-party libraries, it provides richer and more fine-grained recommendation results for the third-party libraries. The inputs of KG2Lib are the interaction matrix $\mathbf{Y}$, the weight matrix $\mathbf{W}$ and the knowledge graph $\mathbf{G}$. The output of the model is the top-N recommendation results of third-party libraries. Figure 1 shows the flow of recommendation.

Our method is divided into two parts: data processing and model processing. The data processing part is mainly composed of four steps: data acquisition, interactive file processing, weight matrix processing and graph file processing. Three datasets are adopted in our experiment. For the first dataset, the projects and libraries were obtained from GitHub. The other two datasets are obtained from our comparative

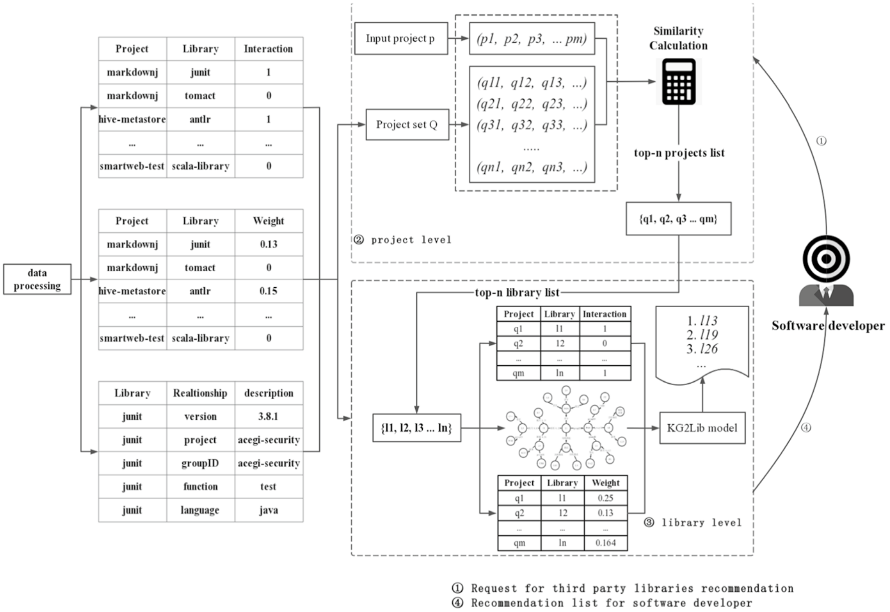

Fig. 1 Overall recommendation framework

baseline methods. According to the calling relationships between the projects and libraries in above three datasets, interactive files and weight matrix files are generated. The knowledge graph of the third-party libraries is constructed according to the relationship data of the third-party libraries (such as version, groupID and language).

The model processing part includes two parts. One is calculating the similarity of the projects from project level and obtain a third-party library set which the libraries are called by similar projects. From third-party library level, fine-grained third-party library information is obtained from the knowledge graph **G**. Then, the vector representation of the third-party libraries is calculated. The interaction matrix **Y**, knowledge graph **G**, and weight matrix **W** are input into the improved GCN for training. The prediction function is in formula (1).

$$\hat{y}_a(p.l) = \tau(p, l|k, Y, W, G) \tag{1}$$

where $\hat{y}_a(p.l)$ is the probability that project p calls the library l, τ represents the KG2Lib function, represents the trainable parameter of KG2Lib.

3.1 Input preprocessing

Interaction matrix Three elements of the recommendation system are users, items and scores (Noia et al., 2021) [41]. Generally, the user–item interaction matrix is used to represent the potential relationship between users and items [42]. Each

Table 1 Interaction matrix Y

Program	Library	Interaction
p_1	l_1	1
p_1	l_2	1
p_1	l_3	0
p_1	l_4	0
p_1	l_5	1
p_2	l_1	1
…	…	…
p_3	l_5	0

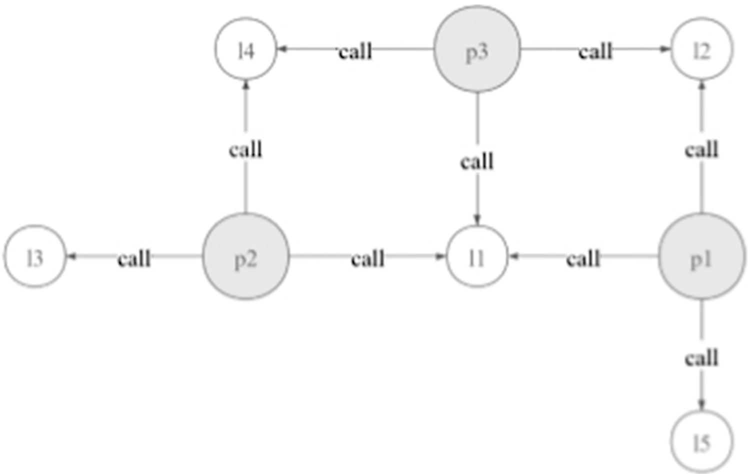

Fig. 2 Relation network example

column in the interaction matrix represents an item, and each row represents a user. The junction of the row an d the column indicates the user's score on the corresponding item [43]. Phung et al. (2013) applied the idea of user–item interaction matrix to the third-party library field. They transformed the relationship between users and items into the interaction between projects and libraries. Each software project may contain multiple third-party libraries. If a project calls a third-party library, the junction value is represented by 1; otherwise, take the value 0. In this paper, the interaction matrix is processed into the form of "Project-library-Label," as shown in Fig. 1. Its description is as follows:

> Given a project set $P = \{p_1, p_2, \ldots, p_m\}$, where m represents the number of the projects; a third-party library set $L = \{l_1, l_2, \ldots, l_n\}$, where n represents the number of third-party libraries;
> Project library interaction matrix $\mathbf{Y} \in R^{m*n}$, which is defined according to the calling relationship between the projects and the libraries, where $y_{pl} = 1$ (y_{pl} indicates

whether the current project p has called library l) indicates that the project p has called the library l, and accordingly, $y_{pl}=0$ indicates that the project p has not called the library l.

For example, given a project set $P=\{p_1, p_2, p_3\}$, a library set $L=\{l_1, l_2, l_3, l_4, l_5\}$. If p_1 calls l_1, l_2, l_5, p_2 calls l_1, l_3, l_4, p_3 calls l_1, l_2, l_4, then their corresponding interaction matrix is expressed in the form shown in Table 1.

According to the interaction of the above projects and libraries, a relationship network between projects and libraries is constructed. The interaction matrix in Table 1 is transformed into the following network in Fig. 2.

Weight matrix The weight matrix is built according to the number of times a project calls a library, so that the third-party libraries can be recommended to developers in a more fine-grained way. The calculation formula is as follows:

$$W_{l_i} = \frac{The\ number\ of\ l_i\ in\ current\ project}{Total\ number\ of\ libraries\ in\ current\ project} \tag{2}$$

Given a project set $P=\{p_1, p_2, p_3\}$, a third-party library set $L=\{l_1, l_2, l_3, l_4\}$, project $p_1 \ni l_1, l_2, l_3$, where l_1 is called twice, l_2 is called three times, l_3 is called three times; project $p_2 \ni l_1, l_3, l_4$, where l_1 is called once, l_3 is called three times, l_4 is called four times; and project $p_3 \ni l_1, l_3$, where l_1 is called twice, l_3 is called once. The total number of libraries in p_1 is 8 and the number of l_1 called by p_1 is 2. According to the above formula, the weight of l_1 in p_1 is 0.25. The others' calculation process is the same as p_1 and l_1. Then, all libraries' weight value in different projects are obtained. Their corresponding weight matrix is shown in Table 2.

Knowledge graph. A knowledge graph is composed of triples. A triple in the knowledge graph is (h, r, t), where h represents the head entity of a triplet, t represents the tail entity of a triplet, and r represents the relation. For example, for the library Junit, the constructed triples (Junit, Junit.Version, 3.8.1) represent that the version of Junit is 3.8.1. Based on the feature of the dataset we obtained, the determined relation types in this paper include library version, function description, groupID and language. Given a project set $P=\{p_1, p_2, p_3\}$, and a third-party library set $L=\{l_1, l_2, l_3, l_4\}$, where

l_1: {version: 1.1.0; function: test; groupID: *PAM*; Language: Java};

Table 2 Weight matrix

Projects	Library	Weight
p_1	l_1	0.25
p_1	l_2	0.375
p_1	l_3	0.375
p_1	l_4	0
p_2	l_1	0.125
…	…	…
p_3	l_3	0.33
p_3	l_4	0

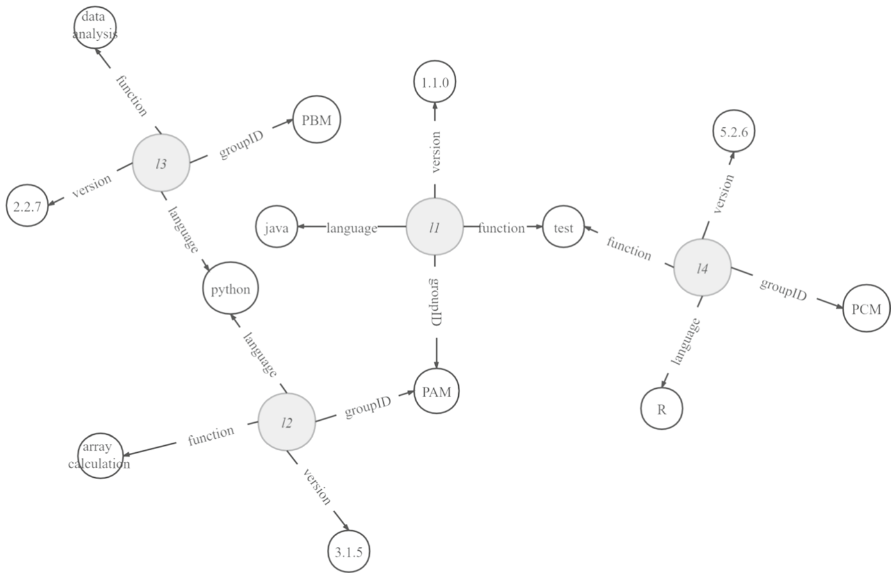

Fig. 3 An example of knowledge graph for third-party libraries

l_2: {version: 3.1.5; function: array calculation; groupID: PAM; Language: Python};

l_3: {version: 2.2.7; function: data analysis; groupID: PBM; Language: Python};

l_4: {version: 5.2.6; function: test; groupID: PCM; Language: R}.

According to the above information, we can get the triple (take the information of l_1 as an example): (l_1, version,1.1.0), (l_1, function, test), (l_1, groupID, PAM), (l_1, Language, Java). The others are the same as l_1. Based on this, a knowledge graph for these third-party libraries is shown in Fig. 3:

3.2 KG2Lib model

Given the project–library interaction matrix **Y**, the weight matrix **W** as well as the knowledge graph **G**, we aim to predict whether library *l* can be recommended to project *p* as the most suitable library. KG2Lib integrates the data information of the knowledge graph **G**, the interaction matrix **Y** and the weight matrix **W**, input into graph convolution network. The visualized representation of the integrated information is shown in Fig. 4.

The core ideas are as follows.

According to the similarity of libraries, the top-N library set to be recommended is obtained, and the library set to be recommended contained in the top-N item is obtained. For these third-party libraries to be recommended, obtain their interaction information with the projects, the weight information in the project and the characteristic information contained in these third-party libraries. For

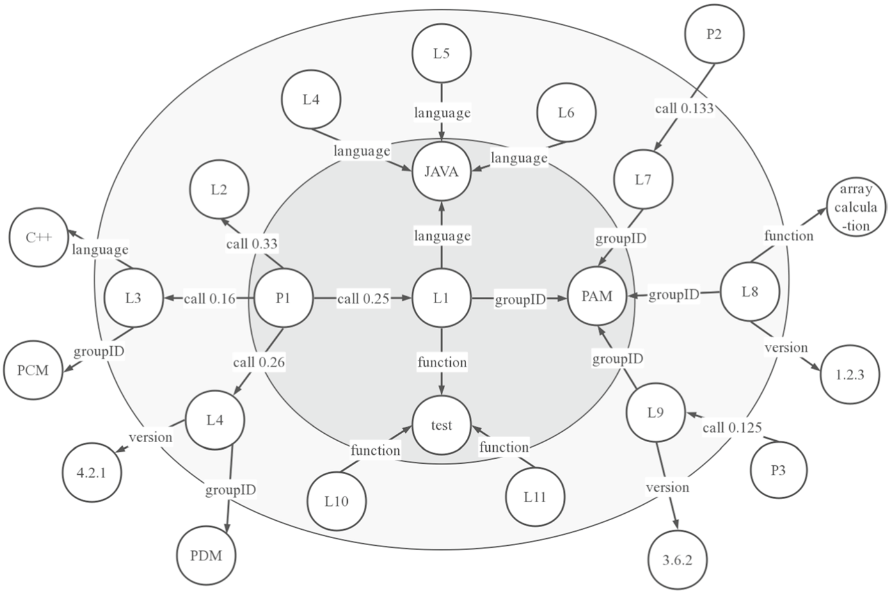

Fig. 4 Visualized representation of GCN input

each third-party library to be recommended, take the current third-party library as the center and calculate its characteristics for the nodes in its neighborhood.

Firstly, the vector representation of the projects in the knowledge graph is calculated. The implementation process is as follows: for a given project p, there are neighbor nodes (l_1, l_2, l_3..., l_n), the characteristics of p can be expressed as $\vec{\delta}_p = \left(\delta_1, \delta_2, \delta_3, \dots, \delta_n\right)$, δ_i represents the weight of the ith library, and its calculation formula is:

$$\delta_i = \frac{\mathcal{F}_i}{\mathcal{A}_i} \tag{3}$$

$\mathcal{F}_i$ is the number of times the ith library appears in project p, $\mathcal{A}_l$ represents the total number of occurrences of n libraries in project p.

The most similar projects can be obtained by calculating the similarity according to the feature vector of the projects. $\vec{\delta}_p = \left(\delta_1, \delta_2, \delta_3 \dots \delta_n\right)$, $\vec{\tau}_q = \left(\tau_1, \tau_2, \tau_3 \dots \tau_m\right)$, the similarity calculation of project p and project q is as follows:

$$\mathrm{simp,q} = \frac{\sum_{a=1}^{k} \delta_a \times \tau_a}{\sqrt{\sum_{a=1}^{k} (\delta_a)^2} \times \sqrt{\sum_{a=1}^{k} (\tau_a)^2}} \tag{4}$$

where k represents the number of third-party libraries shared by p and q [41].

According to the above similarity calculation method, the top-n projects set that most similar to p is obtained. Then, the set of third-party libraries recommended to project p can be obtained. However, this is only recommended third-party libraries to users at the project level, which needs to be extended to the third-party library level more finely to recommend more accurate third-party libraries. By constructing the library knowledge graph, the correlation among libraries can be considered, and the abundant auxiliary information about libraries are used to alleviate the cold start problem. Therefore, the feature vector of the third-party libraries needs to be considered.

A software project may call multiple third-party libraries, and each third-party library is of different importance to the software project. According to the different functions of the software project, the types and emphases of the called third-party libraries are also different, and each third-party library contains different relationship attributes. Therefore, in the recommendation process of a third-party library, calculating the importance of the relationship is very important for the software project. This is useful for recommending the libraries which has high correlation with the project, rather the most popular libraries, which solves the "long tail problem." In this paper, for each recommended library l_i, its feature can be represented as:

$$F_{l_i} = \sum_{ent \in N(l_i)} N(l_i) \mathcal{H}_{proj,rela_{l_i,ent}} \boldsymbol{ent} \tag{5}$$

F_{l_i} represents the feature of l_i, $N(l_i)$ the neighbor set which has relation with l_i, *ent* is the entities set which have link with l_i and $\boldsymbol{ent}$ means the vector representation of *ent*.

$\mathcal{H}_{proj,rela_{l_i,ent}}$ indicates the relevance degree of project and library l_i. Its calculation are as follows:

$$\mathcal{H}_{proj,rela} = h(\boldsymbol{proj}, \boldsymbol{rela}) \tag{6}$$

However, in the actually calculation process, some projects call limited numbers of software libraries, which limit available information and libraries feature they have called. While those projects who called more libraries can get rich feature information and libraries information they have called. Therefore, the libraries' feature should be standardized according to unified standards. In this paper, the standardize libraries feature are expressed as:

$$\mathcal{H}_{proj,rela_{l_i,ent}} = \frac{\exp\left(\mathcal{H}_{proj,rela_{l_i,ent}}\right)}{\sum_{ent \in N(l_i)} \exp\left(\mathcal{H}_{proj,rela_{l_i,ent}}\right)} \tag{7}$$

Meanwhile, we aggregated the projects and the relations in the neighborhood of libraries l_i. The aggregation formula is shown in formula (8).

$$aggr = \sigma\left(w \cdot \left(l_i + \mathcal{H}_{proj,rela_{l_i,ent}}\right) + b\right) \tag{8}$$

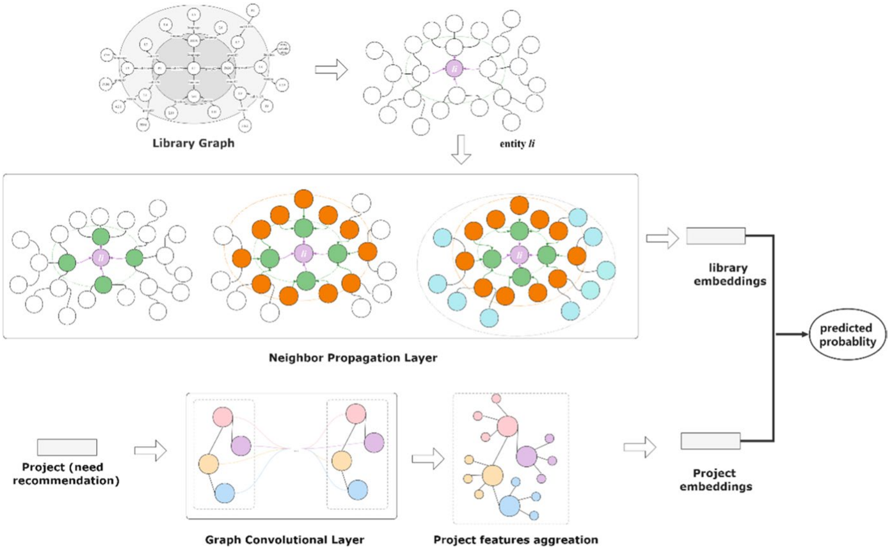

Fig. 5 Process flow of knowledge-graph-based GCN

where w represents the learnable weight, b represents the learnable bias term, σ indicates the ReLU activation function, which completes the aggregation operation. The probability of whether *project* will call the library l_i is in formula (9).

$$\hat{y}_a\left(project \cdot l_i\right) = \ell(\boldsymbol{pro}, \boldsymbol{aggr}) \tag{9}$$

ℓ is the sigmoid activation function. ***pro*** is the representation of a specified project. ***aggr*** is the aggregation feature.

According to the prediction results, it is further filtered on the original set of third-party libraries to be recommended and recommended to developers. The process flow of knowledge-graph-based GCN is shown in Fig. 5.

3.3 Experimental evaluation

Based on the method described in Sect. 3, this section describes the details of the model implementation. Four third-party library recommendation methods, LibRec [4], LibFinder [20], LibCUP (EMASA et al., 2018) and CrossRec [39] which are related to our work, were chosen as our comparative baseline models. To better compare the performance of our method and these four methods, we adopted their datasets as our experimental datasets.

Table 3 Form of datasets

#	Library	Version	GroupID	Function	Language	∈ Project
1	l_1	3.5.2	PAM	Test	Java	p_1
2	l_2	1.2.0	PBM	Array calculation	Python	p_1
…	…	…	…	…	…	…

3.4 Dataset processing

The first data set (dataset1) is obtained from GitHub. It is used to evaluate the performance of KG2Lib and the comparison of KG2Lib, CrossRec and LibRec. The second dataset (dataset2) is derived from the public data set of LibFinder. It is applied for comparing the performance of KG2Lib, CrossRec and LibFinder. The third dataset (dataset3) comes from the public dataset of LibCUP. It is used as the comparison dataset of KG2Lib, CrossRec and LibCUP. The filtering conditions of the original dataset are as follows:

1. Pom.xml file shall be included in the project.
2. The source code of the project exceeds 1500 lines.
3. The number of libraries called by the project exceeds 15.

The form of the processed datasets is shown in Table 3.

3.5 Model implementation

Our data preprocessing algorithm is shown in Algorithm 1. This algorithm only shows the processing flow of weight matrix and interaction matrix. Lines 2–10 is used for calculating the call relationship between the projects and the libraries and storing it in the mapping dictionary. Firstly, the duplicate projects and libraries are removed by the project name. Then, sort the libraries and projects into *Data_P* and *Data_Lib*. For each project p_i, obtain the libraries it called and calculate the number of each library called by p_i, storing them into *List_Lib* and *AllCount_l*, respectively. Algorithm 11–27 calculates the interaction matrix $\mathbf{Y}$ and weight matrix $\mathbf{W}$. For each project p_i, if it has called l_j in *Data_Lib*, the output interaction relation can be expressed as $(p_i, l_j, 1)$, and the weight relation can be expressed as $(p_i, l_j, weightvalue)$. Otherwise, both the interaction relation and the weight relation are $(p_i, l_j, 0)$. The time complexity of algorithm 1 is $O(n^2)$.

Algorithm 1 Data pre-processing

```
Input: Dataset of third-party libraries D
Output: Interaction matrix Y, Weight matrix W
Initialization: Temp interaction matrix Y_ temp;
                Temp weight Matrix W_temp
```

```
function MatrixGeneration(D)
  Projects extraction and de-duplicate → Data_P
  Libraries extraction and de-duplicate → Data_Lib
  Dict_p_lib //The dictionary about projects and libraries
    for p in Data_P:
        store p in Data_p    //obtain the data about project p and store in Data_p
          if lib in Data_p:
              store lib in List_Lib   //obtain the libraries in Data_p and store in List_Lib
      AllCount_l   // obtain all counts of libraries in p
      Dict_p_lib. append{(p: List_Lib)}
  end for
Y_temp  //new temporary interaction matrix
W_temp  //new temporary weight matrix
  for p in Data_P:
    for l in Data_Lib:
      if l in p_lib:          //if p called l
        Y_temp = [p, l, 1]   // the interaction information of p and l is 1
        count(p, l)          //calculate the counts that p calls l
        weight (p, l) = count(p, l)/ AllCount_l
            Y_temp = [p, l, weight (p, l)]
         end if
         else:
           Y_temp = [p, l, 0]
           W_temp = [p, l, 0]
        end else
    end for
  end for
```

The algorithm for generating candidate project is shown in Algorithm 2. Lines 2–12 of the algorithm first calculate the feature vector and similarity of project p and project set Q. For any p (the projects that need to be recommended libraries), calculate its vector representation F_p. Then, for each project q in set Q, calculate its feature vector F_q. Last, the similarity between p and q can be obtained. Then, sort the projects to be recommended according to the similarity values. After that, the top-N project list that will be recommended is obtained. The time complexity of Algorithm 2 is $O(n)$.

Algorithm 2 Candidate project set calculation

Input: project (need recommend) p, project set (need to be recommend) Q
Output: candidate project set *Cset*
Initialization: temp project set *Qset*,
libraries set which called by *m set_m*,
feature vector F_i,
weight value for l_i weight_i

*function LibSetGeneration(**L**)*
set_p= $\{l_1, l_2, l_3, \ldots, l_a\}$
for l_i **in** *set_p:*
  $weight\_i = the\ number\ of\ li\ in\ p/a$
  $F_p \leftarrow weight\_i$ *// calculating the feature of p*
 for *q* **in** Q:
   set_q $=\{ l_1, l_2, l_3, \ldots, l_b \}$
   for l_i **in** *set_q:*
   $weight\_i = the\ number\ of\ li\ in\ q/b$
   $F_q \leftarrow weight\_i$ *// calculating the feature of q*
   sim(F_p, F_q) *// calculating the similarity of p and q*
sort sim(F_p, F_q)→top-n *Qset* *//generate top-n list according to similarity value*

3.6 Evaluation metrics

To evaluate the method proposed in this paper, the below indicators were adopted. These indicators are often used in the evaluation of recommendation systems in the field of software engineering (Tran et al., 2021) [44]. Firstly, the meaning of the following symbols is introduced:

N: The cut-off value of the ranking list. For example, N = 5 means that the top five recommendations are selected.
k: The number of neighbor entities applied to the recommendation process.

1. Success rate

Success rate refers to a given set of projects to be tested, $P = \{p_1, p_2, p_3, \ldots, p_n\}$, for any $p_i \in P, (i = 1, 2, 3, \ldots n)$, a recommendation system returns at least one library [4]. Its calculation formula is:

$$successrate = \frac{count_{p \in P}(\left|result_N(p)\right| > 0)}{result_{all}} \tag{10}$$

where count () represents the number of times the Boolean expression specified in its parameter is true. $count_{p \in P}(|result_N(p)| > 0)$ is the number that the recommendation results are not null. $result_{all}$ indicates the total number of recommendations.

2. Recommendation diversity

Recommendation diversity refers to the ability of the system to provide developers with as many third-party libraries as possible. The recommended third-party libraries should cover a wider range than only a small number of popular third-party libraries [45]. Its evaluation indicators are coverage degree COV and focus degree FOC, as shown in formulae (11) and (12). COV represents the percentage of libraries recommended to the project in the total number of libraries. FOC is used to evaluate whether the recommended results only focus on a small number of third-party libraries. For a given set of candidate third-party libraries l that can be recommended, $numc(l)$ represents the number of projects calling library l ($l \in L$), and *total* represent the number of recommended third-party libraries in all projects.

$$COV = \frac{\sum_{i=1}^{n} Result(p_i)}{L} \tag{11}$$

$$FOC = -\sum_{l \in L} \left(\frac{\#numc(l)}{total} \right) ln \left(\frac{\#numc(l)}{total} \right) \tag{12}$$

3. Novelty

When recommending a library, novelty is used to measure whether the system can recommend a library to developers from the “long tail library set [46]. In the three data sets used in this paper, most of the third-party libraries are called by the project less times, while some popular libraries are called many times in the project. The evaluation index is EPC (expected population complex) [47] as shown in formula (13).

$$EPC = \frac{\sum_{p \in P} \sum_{m=1}^{N} \frac{deg(p,m) * [1 - pop(Fre_m(p))]}{\log_2(m+1)}}{\sum_{p \in P} \sum_{m=1}^{N} \frac{deg(p,r)}{\log_2(m+1)}} \tag{13}$$

where $deg(p, m)$ indicates the correlation between the library and project p at the m position of the top-N list. If it is relevant, take 1; if it is not relevant, take 0. $pop(Fre_m(p))$ indicates the popularity of the third-party database in the m position of the top-N list. The more unpopular the libraries recommended by the system, the higher the EPC value, and the more novel the recommended method can be reflected.

4. CTR prediction

The evaluation index es of CTR (click through rate) are AUC (area under the curve) and F1 [48]. AUC is defined as the area enclosed by the coordinate axis under the ROC (receiver operating characteristics) curve, and its value range is in the range of 0.5 and 1. The higher the AUC value is, the higher the authenticity of the detection method is. F1 value is evaluated by Precision and Recall, and its calculation formula is as follows:

$$\mathrm{F1} = \frac{2 * Precision * Recall}{(Precision + Recall)} \tag{14}$$

The precision indicates the ratio of libraries which is related to the current projects. Its calculation process is as follows:

$$Precision == \frac{Lib_{Revelant}}{Lib_{AllRec}} \tag{15}$$

$Lib_{Revelant}$ is the number of libraries which is suitable for the current libraries and Lib_{AllRec} is the total number of recommended libraries. *Recall* is the ratio of Lib_{AllRec} and the actual number of libraries, and it can be represented as follows:

$$Recall == \frac{Lib_{AllRec}}{Lib_{Actual}} \tag{16}$$

4 Experimental evaluation

In this section, series of experiments were conducted to verify the performance of KG2Lib. Firstly, the model is experimentally analyzed from two aspects of CTR prediction and top-N recommendation to evaluate the performance of KG2Lib. Then, the recommended performance of four baselines (LibRec [4], LibFinder [20], LibCUP (EMASA et al., 2018) and CrossRec [39] is compared with KG2Lib. Finally, some limitations of KG2Lib were point out.

The evaluation process of the experiments is mainly divided into three stages: data processing, model training and result evaluation. In the data processing stage, we obtain the project file from GitHub website and parses it according to the pom.xml file to form a dataset file, which is used to evaluate KG2Lib, LibRec, CrossRec, LibFinder and LibCUP. Each dataset is divided into three parts: training set, verification set and test set. Their ratio is 6:2:2. In the evaluation stage, we compare the recommended results with the data in the validation set to calculate the evaluation index.

Meanwhile, to prove the effectiveness of KG2Lib, we make statistics on the calling frequency of third-party libraries in three datasets. The calling frequency are divided into five intervals, namely "< 10," "11–20," "21–50," "51–200" and "> 200." Among the 13,497 third-party libraries contained in dataset1, 12,962

Table 4 Distribution of libraries in datasets

Projects	< 10	11–20	21–50	54–200	> 200
Libraries (dataset1)	**12,962**	280	164	81	**10**
Libraries (dataset2)	**3275**	600	664	432	**158**
Libraries (dataset3)	**47,999**	3236	1974	1110	**246**

Table 5 Success rate@5 of KG2Lib, CrossRec and LibRec on dataset1

Model	k = 5	k = 10	k = 15	k = 20	k = 25
LibRec	0.876	0.862	0.868	0.863	0.868
CrossRec	0.903	0.931	0.929	0.926	0.929
KG2Lib	**0.924**	**0.932**	**0.931**	**0.928**	**0.933**

third-party libraries (about 96.03% of the total number of third-party libraries) were called less than 10 times, and 91 third-party libraries (about 0.6% of the total number) were called more than 50 times. Among the 5129 third-party libraries contained in dataset2, 3275 third-party libraries (accounting for 63% of the total number of third-party libraries) were called less than 10 times, and 590 third-party libraries (accounting for about 12% of the total number) were called more than 50 times. Among the 56,365 third-party libraries contained in dataset3, 49,799 third-party libraries (accounting for about 88% of the total number of third-party libraries) were called less than 10 times, and 1356 (accounting for about 2% of the total number of third-party libraries) were called more than 50 times. The bold style of the data in the table are to showing the long tail problem in sofeware libraries recommendation, from the table we can see that: the three datasets have a common feature: the vast majority of third-party libraries are called less, and a few third-party libraries are called frequently. This is also the feature of the recommendation field, that is, the "long tail problem" (Table 4).

4.1 Experimental comparison on dataset1

We compared dataset1 with LibRec, CrossRec and KG2Lib by using different combinations of number of recommended libraries (N) and number of neighbor projects exploited in the recommendation phase (k). Varying N means changing the length of the recommendation list, whereas increasing k means considering more neighbor projects for the recommendation.

The experimental results of KG2Lib, CrossRec and LibRec on the success rate are shown Tables 5, 6, 7, and 8, the bold style of the result in table shows that in

Table 6 Success rate@10 of KG2Lib, CrossRec and LibRec on dataset1

Model	k = 5	k = 10	k = 15	k = 20	k = 25
LibRec	0.864	0.864	0.867	0.865	0.863
CrossRec	0.945	0.956	0.950	0.954	0.955
KG2Lib	**0.95**	**0.955**	**0.956**	**0.958**	**0.961**

Table 7 Success rate@{1,3,5,7,10}, k = 10 of KG2Lib, CrossRec and LibRec on dataset1

Model	N = 1	N = 3	N = 5	N = 7	N = 10
LibRec	0.647	0.813	0.865	0.901	0.925
CrossRec	0.697	0.879	**0.919**	0.939	0.956
KG2Lib	**0.713**	**0.890**	0.910	**0.942**	**0.964**

Table 8 Success rate@{1,3,5,7,10}, k = 20 of LibRec, CrossRec and KG2Lib on dataset1

Model	N = 1	N = 3	N = 5	N = 7	N = 10
LibRec	0.673	0.819	0.868	0.896	0.925
CrossRec	0.736	0.881	0.924	0.937	0.953
KG2Lib	**0.754**	**0.892**	**0.926**	**0.941**	**0.954**

differentt indicators and conditions, the method who have achieved the best result. Four groups of experiments were conducted on dataset1, respectively, changing the number of recommended projects and the number of neighbor projects included in the recommended range. Table 5 shows that when the number of recommended third-party libraries is 5, changing the k value from 5 to 25, the success rate of KG2Lib is always higher than that of CrossRec and LibRec. It can be seen that at the condition of success rate@5, the maximum success rate of LibRec and CrossRec is 0.876 and 0.903, respectively, while the maximum success rate of KG2Lib is 0.933. This shows that KG2Lib gets better performance than the other two approaches.

When the number of recommended projects is 10, it can be seen that the success rate of LibRec, CrossRec and KG2Lib changes slightly when altering the value of k. However, the value of KG2Lib is still higher than that of LibRec and CrossRec. It can be seen from Tables 5 and 6 that when the recommended list value remains unchanged, the success rate changes little by changing the value of k.

Next, we investigate the success rate with regard to N. We consider a small number of recommended items, i.e., N = {1, 3, 5, 7, 10}. In practice, this means that the developer wants to see a short list of recommended libraries. In the first experiment, keeping k = 10 and change the value of N (the length of the recommendation list), the outcomes are described in Table 7. When N = 1, the success rates of LibRec, CrossRec and KG2Lib are 0.647, 0.697 and 0.713, respectively. That is, if the current user only needs one search result, KG2Lib can also obtain higher success power than LibRec and CrossRec. On the other hand, the performance of this method is also verified. However, we found that when k = 10, N = 5, the success rate of KG2Lib is 0.910, which is less than CrossRec. But on the whole, the overall success rate of KG2Lib are higher than the other two methods.

When k = 20 and N = 1, we found that the success of three methods are all improved, indicating that expanding the number of recommended projects can improve the success rate of the model. In addition, when changing N from 1 to 10, the success rate values of KG2Lib are always higher than LibRec and CrossRec. To further observe this phenomenon, we conducted more experiments with an increasing k, e.g., k = {50, 60, 100}. As far as we can see, there are no subtle differences

Table 9 AUC and F1 of KG2Lib on dataset1

Dataset	AUC	F1
Dataset1	0.975	0.927
Dataset2	0.852	0.813
Dataset3	0.699	0.674

Table 10 Success rate of CrossRec, LibFinder and KG2Lib on dataset2

Model	N					
	1	2	4	6	8	10
LibFinder	0.633	0.698	0.813	**0.876**	**0.904**	**0.918**
CrossRec	0.771	0.816	0.851	0.860	0.863	0.864
KG2Lib	**0.779**	**0.821**	**0.857**	0.872	0.876	0.881

between the conclusions obtained from the new experiments with those previously presented in the paper. Thus, for the sake of clarity, the outcomes of these experiments are omitted from the paper (Table 9).

In addition, we analyze the CTR performance of KG2Lib on dataset1. The results are as follows:

It can be seen from the results above that the AUC and F1 value in three datasets descended gradually. The dataset1 contains 1200 projects and 13,497 libraries, the dataset2 contains 29,653 projects and 5129 libraries, while the dataset3 contains 90,475 projects and 56,435 libraries, which means that the CTR performance of KG2Lib are getting smaller with the increase in data set size.

4.2 Experimental comparison on dataset2

We compared dataset2 with LibFinder, CrossRec and KG2Lib by using different combinations of number of recommended libraries (N) varying N means changing the length of the recommendation list, whereas increasing k means considering more neighbor projects for the recommendation. The success rate of CrossRec, LibFinder and KG2Lib on dataset2 is shown in Table 10 the bold style of the result in table shows that in different indicators and conditions, the method who have achieved the best result.

When N = {1,2,4}, the success rate of KG2Lib is higher than that of CrossRec and LibFinder, while when N = {6,8,10}, the success rate of KG2Lib is higher than CrossRec, but lower than LibFinder. LibFinder is more advantageous when developers want to get more recommended library results. This is also a direction that our work need to improve in the future.

In addition, when preprocessing dataset2, some projects call only one or two libraries. Phung et al. (2020) believed that such metadata pairs will reduce the recommended performance of the model, so the project to be tested should contain more libraries. If the method heavily depends on the historical interactive data of

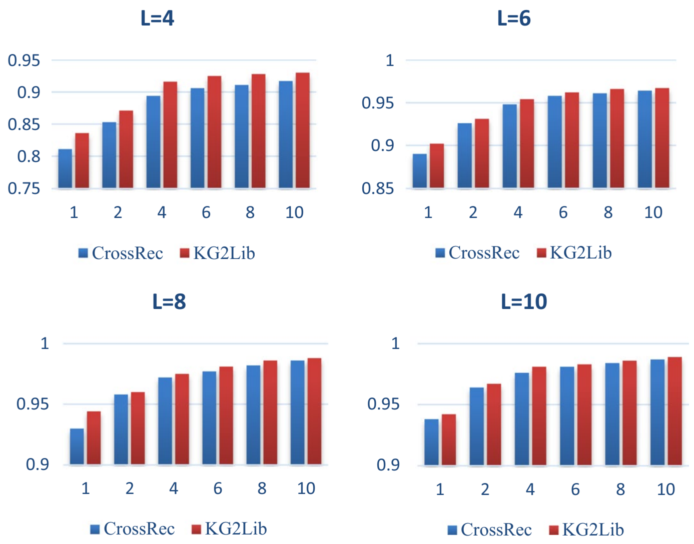

Fig. 6 Success rate of CrossRec and KG2Lib on dataset2 with different number of libraries

the projects and libraries, there will be a cold start problem. Based on this problem, we use the relevant entity information in the knowledge graph of the third-party libraries to improve the recommendation performance of the model. In this paper, the experimental settings of Phung et al. (2016) are used to screen the projects and select the project sets which has called 4, 6, 8 and 10 third-party libraries, respectively. The experimental comparison results of KG2Lib and CrossRec on dataset2 is shown in Fig. 5.

Figure 6 shows the success rate obtained by varying L. As it can be seen, KG2Lib's performance is proportional to L; the more densely (with respect to dependencies) the projects are, the better performance KG2Lib achieves. For instance, when only considering projects with at least 4 libraries, i.e., L = 4, the success rate obtained for N = 10 is 0.93. However, if L is increased to 10, the corresponding success rate improves to reach 0.989. The same trend can be witnessed by other combinations of L and N.

4.3 Experimental comparison on dataset3

We compared dataset3 with LibCUP, CrossRec and KG2Lib by using different combinations of number of recommended libraries (N) varying N means changing the length of the recommendation list, whereas increasing k means considering more neighbor projects for the recommendation. The success rate of CrossRec, LibFinder and KG2Lib on dataset3 is shown in Fig. 7.

When the cut-off value of the recommendation list is 1, the value of LibCUP is 0.12, the success rate of CrossRec is 0.21, and the success rate of KG2Lib is 0.23. For other values of N, the performance of the method in this paper is also higher than that of LibCUP and CrossRec. When developers hope to get ten libraries, the

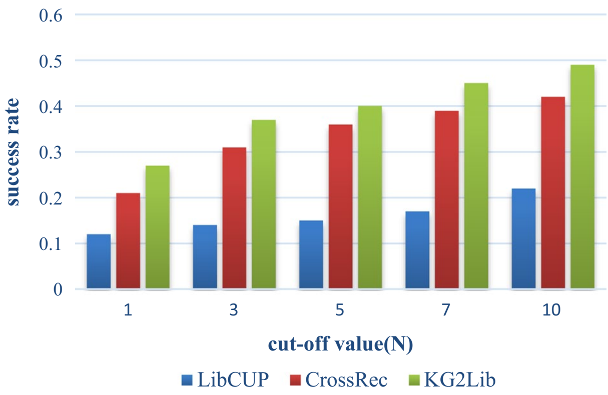

Fig. 7 Success rate of LibCUP, CrossRec and KG2Lib on dataset3

Table 11 COV for N = {5, 15, 25}, k = {10, 20}

N	K = 10			K = 20		
	LibRec	CrossRec	KG2Lib	LibRec	CrossRec	**KG2Lib**
5	0.857	1.099	1.326	0.691	0.814	**0.873**
15	2.675	3.278	3.632	1.937	2.312	**2.255**
25	4.594	5.897	5.969	3.139	4.005	**4.132**

success rate of LibCUP, CrossRec and KG2Lib is 0.22, 0.42 and 0.49, respectively. However, the success rate value is lower when compared with dataset2 and dataset1. The reason is the scale of dataset3 is larger than the other two, which indicates that with the increase in the dataset, the data sparsity problem has a negative influence on recommendation model. This phenomenon is also the future research direction of our work. But, according to the current experimental results, we can see that the result tendency of KG2Lib is basically consistent with the other two datasets.

4.4 Other experiments

In addition, we evaluated the performance of LibRec, CrossRec and KG2Lib from three aspects: coverage, entropy and novelty. In the three sets of experimental settings, the number of recommended results is N = {5, 15, 25}. The number of projects included in neighbors is k = 10 and 20, respectively. According to the recommended diversity formula, the recommended diversity is determined by the value of coverage and entropy. The higher the value of coverage, the lower the entropy, indicating that the third-party libraries recommended by the model is not limited to a specific group of popular libraries. From the exp erimental results of various indicators in Tables 11 and 12, the diversity value of KG2Lib is significantly higher than that of LibRec and CrossRec. It shows that the recommended results of the proposed method can cover more types of third-party libraries. The bold style of

Table 12 FOC for N = {5, 15, 25}, k = {10, 20}

N	K = 10			K = 20		
	LibRec	CrossRec	KG2Lib	LibRec	CrossRec	KG2Lib
5	0.869	0.239	**0.21**	0.552	0.127	**0.119**
15	2.653	0.723	**0.665**	1.639	0.381	**0.342**
25	4.500	1.271	**1.118**	2.751	0.635	**0.593**

tha result in table shows that in differentt indicators and conditions, the method who have achieved the best result.

EPC is an evaluation index used to evaluate the novelty of the model. A higher EPC value indicates that the recommended third-party libraries are from “long tail library set”, which are not so popular but highly correlated with the projects. By comparing KG2Lib with the other two methods, Table 13 shows that the EPC value of KG2L ib is higher than LibRec and CrossRec, the bold style of the result in table shows that in differentt indicators and conditions, the method who have achieved the best result. However, it can be found from the data in the observation table that although the EPC value of KG2Lib is higher than the other two methods, it is not much improved; this is also a direction for our future research work.

5 Conclusions

Software third-party libraries are important reusable software resources in software development activities. Calling third-party libraries can reduce the workload of developers and improve the software development efficiency. In this paper, we introduce KG2Lib, a graph convolutional network for third-party library recommendation. KG2Lib introduces the knowledge graph to represent the relationship between projects and libraries and enrich data information of the third-party libraries. The vector representation of projects in the graph is obtained from the project level, and the similarity between projects is calculated. According to the similarity values, the top-N projects and the third-party libraries called by the projects are obtained, and the collection of libraries to be recommended is constructed. Then, considering whether a library in the library set can be recommended, the improved graph convolution network is adopted to learn the characteristics of the third-party libraries in the graph, and recommend the third-party libraries to users more finely. The experimental results verify the performance of the KG2Lib method proposed in this paper. The methods proposed in this paper are better than the other current methods.

For the future work, more related information should be obtained to enrich the library knowledge graph to obtain more diversified recommendation results. In addition, when developers expect to get more recommendation results, the success rate of the method in this paper is lower than that of LibFinder. This requires us to further improve the success rate of KG2Lib in recommending more results.

Acknowledgements This work is supported by the National Natural Science Foundation of China under Grant Nos. 61862063, 61502413, 61262025; the National Social Science Foundation of China under

Table 13 EPC for N = {5, 15, 25}, k = {10, 20}

N	K = 10			K = 20		
	LibRec	CrossRec	KG2Lib	LibRec	CrossRec	KG2Lib
5	0.187	0.291	0.311	0.114	0.292	**0.302**
15	0.296	0.376	0.388	0.204	0.377	**0.381**
25	0.349	0.401	0.416	0.261	0.416	**0.421**

Grant No. 18BJL104; the Science Foundation of Young and Middle-aged Academic and Technical Leaders of Yunnan under Grant No. 202205AC160040; the Science Foundation of Yunnan Jinzhi Expert Workstation under Grant No. 202205AF150006; the Natural Science Foundation of Key Laboratory of Software Engineering of Yunnan Province under Grant No. 2020SE301; the Science Foundation of "Knowledge-driven intelligent software engineering innovation team."

Data availability statement All the datasets analyzed during the current study are available from the corresponding author on reasonable request.

References

1. Alnusair A, Rawashdeh M, Alhamid MF, Hossain MA, Muhammad G (2016) Reusing software libraries using semantic graphs. In: 2016 IEEE 17th international Conference on information reuse and integration (IRI), pp 531–540. IEEE. doi: https://doi.org/10.1109/IRI.2016.79
2. Yang F, Hong M, Li K (1999) Software reuse and software component technology. Acta Electronica Sinic A
3. Bauer V, Heinemann L, Deissenboeck F (2012) A structured approach to assess third-party library usage. IEEE. https://doi.org/10.1109/ICSM.2012.6405311
4. Thung F, Lo D, Lawall J (2013) Automated library recommendation. Reverse Engineering. IEEE
5. Nagarnaik P, Thomas A (2015) Survey on recommendation system methods. In: 2015 2nd international Conference on electronics and communication systems (ICECS). IEEE, 2015, pp 1603–1608
6. Wang X, Liu X, Liu J, Chen X, Wu H (2021) A novel knowledge graph embedding based API recommendation method for Mashup development. World Wide Web 24(3):869–894. https://doi.org/10.1007/s11280-021-00894-3
7. Null LI, Han N (2021) A time-aware hybrid recommendation scheme combining content-based and collaborative filtering. Front Comput Sci. https://doi.org/10.1007/s11704-020-0028-7
8. Chen J, Yu J, Lu W, Qian Y, Li P (2021) IR-Rec: an interpretive rules-guided recommendation over knowledge graph. Inf Sci 563:326–341. https://doi.org/10.1016/j.ins.2021.03.004
9. Pan H, Yang X (2021) Intelligent recommendation method integrating knowledge graph and Bayesian network. Soft Comput, pp 1–10. doi:https://doi.org/10.1007/s00500-021-05735-z
10. Yang Z, Dong S (2020) HAGERec: hierarchical attention graph convolutional network incorporating knowledge graph for explainable recommendation. Knowl-Based Syst 204:106194. https://doi.org/10.1016/j.knosys.2020.106194
11. Ohtomo K, Harakawa R, Ogawa T, Haseyama M, Iwahashi M (2021) Personalized recommendation of tumblr posts using graph convolutional networks with preference-aware multimodal features. ITE Trans Media Technol Appl 9(1):54–61. https://doi.org/10.3169/mta.9.54
12. Zhong T, Zhang S, Zhou F, Zhang K, Trajcevski G, Wu J (2020) Hybrid graph convolutional networks with multi-head attention for location recommendation. World Wide Web 23(6):3125–3151. https://doi.org/10.1007/s11280-020-00824-9
13. Zheng Y, Gao C, He X, Li Y, Jin D (2020) Price-aware recommendation with graph convolutional networks. In: 2020 IEEE 36th international Conference on data engineering (ICDE). IEEE, pp 133–144

14. Zheng Y, Gao C, He X, Li Y, Jin D (2020a) Price-aware recommendation with graph convolutional networks. In: 2020 IEEE 36th international Conference on data engineering (ICDE). IEEE, pp 133–144
15. Katsuragawa D, Ihara A, Kula RG, Matsumoto K (2018) Maintaining third-party libraries through domain-specific category recommendations. In: 2018 IEEE/ACM 1st international workshop on software health (SoHeal), pp 2–9. IEEE
16. Sun X, Xu C, Li B, Duan Y, Lu X (2019) Enabling feature location for API method recommendation and usage location. IEEE Access 7:49872–49881
17. Sun Z, Liu Y, Cheng Z, Yang C, Che P (2020) Req2Lib: a semantic neural model for software library recommendation. In: 2020 IEEE 27th international Conference on software analysis, evolution and reengineering (SANER), pp 542–546. IEEE. doi:https://doi.org/10.1109/SANER48275.2020.9054865
18. Xu C, Sun X, Li B, Lu X, Guo H (2018) MULAPI: Improving API method recommendation with API usage location. J Syst Softw 142:195–205. https://doi.org/10.1016/j.jss.2018.04.060
19. Deshpande N, Mkaouer MW, Ouni A, Sharma N (2022) Search-based third-party library migration at the method-level. In: International Conference on the applications of evolutionary computation (Part of EvoStar). Springer, Cham, pp 173–190
20. Ouni A, Kula RG, Kessentini M, Ishio T, German DM, Inoue K (2017) Search-based software library recommendation using multi-objective optimization. Inf Softw Technol 83:55–75. https://doi.org/10.1016/j.infsof.2016.11.007
21. Zhao X, Li S, Yu H, Wang Y, Qiu W (2019) Accurate library recommendation using combining collaborative filtering and topic model for mobile development. IEICE Trans Inf Syst 102(3):522–536. https://doi.org/10.1587/transinf.2018EDP7227
22. D'Souza AR, Yang D, Lopes CV (2016) Collective intelligence for smarter API recommendations in python. In: 2016 IEEE 16th international working Conference on source code analysis and manipulation (SCAM). IEEE, pp 51–60
23. Yun W, Zhang X, Li Z, Liu H, Han M (2021) Knowledge modeling: a survey of processes and techniques. Int J Intell Syst 36(4):1686–1720
24. Heiko P (2016) Knowledge graph refinement: a survey of approaches and evaluation methods. Semantic Web 8(3):489–508. https://doi.org/10.3233/SW-160218
25. Wang HM, Nie GH (2007) Research on collaborative filtering algorithm based on fusing user and item's correlative information. J Wuhan Univ Technol
26. Lei R, Gu J, Xia W (2010) An item-based collaborative filtering algorithm utilizing the average rating for items. In: Signal processing & multimedia-international Conferences. DBLP
27. Liu A, Li B (2015) Collaborative filtering algorithm based on the similarity of user ratings and item attributes. In: 2015 3rd international Conference on mechatronics and industrial informatics (ICMII 2015). Atlantis Press, pp 451–455
28. Guan Z (2018) Multi-feature collaborative filtering recommendation for sparse dataset. Springer, Cham
29. Jiang B, Yang J, Qin Y, Wang T, Wang M, Pan W (2021a) A service recommendation algorithm based on knowledge graph and collaborative filtering. IEEE Access 9:50880–50892. https://doi.org/10.1109/ACCESS.2021.3068570
30. Zhang L, Li X, Li W, Zhou H, Bai Q (2021) Context-aware recommendation system using graph-based behaviours analysis. J Syst Sci Syst Eng 30(4):482–494. https://doi.org/10.1007/s11518-021-5499-z
31. Dong B, Zhu Y, Li L, Wu X (2021) Hybrid collaborative recommendation of co-embedded item attributes and graph features. Neurocomputing 442:307–316. https://doi.org/10.1016/j.neucom.2021.01.129
32. Zhang Y, Wang J, Luo J (2020) Knowledge graph embedding based collaborative filtering. IEEE Access, 2020. https://doi.org/10.1109/ACCESS.2020.3011105
33. Yu B, Zhou C, Zhang C, Wang G, Fan Y (2020) A privacy-preserving multi-task framework for knowledge graph enhanced recommendation. IEEE Access 8:115717–115727
34. Dang D, Chen C, Li H, Yan R, Guo Z, Wang X (2021) Deep knowledge-aware framework for web service recommendation. J Supercomput 77(12):14280–14304
35. Zhang Y, Wang J, Luo J (2020) Knowledge graph embedding based collaborative filtering. IEEE Access, 2020
36. Mei D, Huang, Li X (2021) Light graph convolutional collaborative filtering with multi-aspect information. IEEE Access, 2021, doi:https://doi.org/10.1109/ACCESS.2021.3061915

37. Zhang Z, Bu J, Li Z, Yao C, Wang C, Wu J (2021) TigeCMN: on exploration of temporal interaction graph embedding via coupled memory neural networks. Neural Netw 140:13–26. https://doi.org/10.1016/j.neunet.2021.02.016
38. Saied MA, Ouni A, Sahraoui H, Kula RG, Inoue K, Lo D (2018) Improving reusability of software libraries through usage pattern mining. J Syst Softw 145:164–179. https://doi.org/10.1016/j.jss.2018.08.032
39. Nguyen PT, Di Rocco J, Di Ruscio D, Di Penta M (2020) CrossRec: supporting software developers by recommending third-party libraries. J Syst Softw 161:110460
40. Chen J, Li B, Wang J, Zhao Y, Yao L, Xiong Y (2020) Knowledge graph enhanced third-party library recommendation for mobile application development. IEEE Access 8:42436–42446
41. Noia TD, Ostuni VC (2015) Recommender systems and linked open data. In: Proceedings of the 11th international summer school reasoning web. web logic rules, Berlin, Germany, July 31–August 4, 2015, Tutorial Lectures, pp 88–113. doi: https://doi.org/10.1007/978-3-319-21768-0_4
42. Hell F, Taha Y, Hinz G, Heibei S, Müller H, Knoll A (2020) Graph convolutional neural network for a pharmacy cross-selling recommender system. Information 11(11):525. https://doi.org/10.3390/info11110525
43. Yin C, Shi L, Sun R, Wang J (2020) Improved collaborative filtering recommendation algorithm based on differential privacy protection. J Supercomput 76(7):5161–5174
44. Tran DH, Sheng QZ, Zhang WE, Aljubairy A, Zaib M, Hamad SA, Khoa NLD (2021) HeteGraph: graph learning in recommender systems via graph convolutional networks. Neural Comput Appl, pp 1–17
45. Robillard M, Walker R, Zimmermann T (2010) Recommendation systems for software engineering. IEEE Softw 27(4):80–86
46. Jiang Y, Ma H, Liu Y, Li Z, Chang L (2021) Enhancing social recommendation via two-level graph attentional networks. Neurocomputing 449:71–84. https://doi.org/10.1016/j.neucom.2021.03.076
47. Vargas S, Castells P (2014) Improving sales diversity by recommending users to items. In: Proceedings of the eighth ACM Conference on recommender systems, RecSys ’14, Foster City, Silicon Valley, CA, USA–October 06–10, 2014, pp 145–152
48. Blei DM, Ng A, Jordan MI (2003) Latent dirichlet allocation. J Mach Learn Res. https://doi.org/10.1162/jmlr.2003.3.4-5.993