Knowledge Graph based Explainable Question Retrieval for Programming Tasks

Mingwei Liu, Simin Yu, Xin Peng, Xueying Du, Tianyong Yang, Huanjun Xu, and Gaoyang Zhang

***Abstract*—Developers often seek solutions for their programming problems by retrieving existing questions on technical Q&A sites such as Stack Overflow. In many cases, they fail to find relevant questions due to the knowledge gap between the questions and the queries or feel it hard to choose the desired questions from the returned results due to the lack of explanations about the relevance. In this paper, we propose KGXQR, a knowledge graph based explainable question retrieval approach for programming tasks. It uses BERT-based sentence similarity to retrieve candidate Stack Overflow questions that are relevant to a given query. To bridge the knowledge gap and enhance the performance of question retrieval, it constructs a software development related concept knowledge graph and trains a question relevance prediction model to re-rank the candidate questions. The model is trained based on a combined sentence representation of BERT-based sentence embedding and graph-based concept embedding. To help understand the relevance of the returned Stack Overflow questions, KGXQR further generates explanations based on the association paths between the concepts involved in the query and the Stack Overflow questions. The evaluation shows that KGXQR outperforms the baselines in terms of accuracy, recall, MRR, and MAP and the generated explanations help the users to find the desired questions faster and more accurately.**

***Index Terms*—Relevant Question Retrieval, Stack Overflow, Knowledge Graph**

I. Introduction

Technical Q&A sites such as Stack Overflow (SO) play an increasingly important role in software development [1], as evidenced by over 19 million questions and 29 million answers on SO [2]. When facing software programming problems such as implementing specific functionalities or handling errors in code developers often turn to SO for help [1], [3]. Before asking new questions on SO, developers often tend to seek existing questions that are relevant to their needs using the SO search interface or search engines such as Google. However, due to the gap between the queries and SO questions (usually their titles), it is often hard for the developers to find the desired questions and answers efficiently.

Existing researches use information retrieval (IR) and deep learning techniques to retrieve or recommend SO questions that are relevant to a given query [4], [5], [6], [7], [8]. Traditional IR techniques such as TF-IDF [9] and BM25 [10] can efficiently retrieve relevant questions that share common words with the query. However, these approaches fail to address the problem of the lexical gap, *i.e.,* similar or relevant meaning expressed in different words. Recently, some researches [6], [7], [8] use word embedding techniques (*e.g.,* Word2Vec [11] and Glove [12]) to obtain the vector representations of words and sentences and retrieve relevant questions based on the similarities between the vector representations of the questions and the query. Word embedding techniques capture the contexts of words in documents and thus can address the lexical gap between SO questions and queries. However, they cannot capture the semantic differences of different sentence patterns, as they treat a sentence as a bag of words. This shortcoming can be alleviated by using contextual word embedding models such as BERT (Bidirectional Encoder Representations from Transformers) [13]. However, in some cases, there is a knowledge gap between the questions and the query which needs to be bridged by relevant concepts and relations. Moreover, the developers may require explanations about the relevance of the returned SO questions to choose the desired ones.

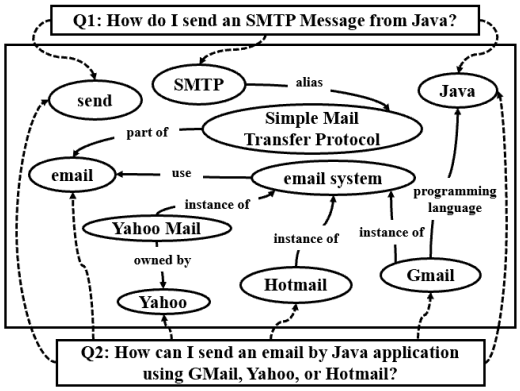

Fig. 1. An Example of Knowledge Gap between Questions

Figure 1 shows an example of the knowledge gap between two SO questions (Q1 [14] and Q2 [15]). Both of them are about sending emails in Java and the accepted answer to one question can work for the other. The rectangles, ellipses, and edges in the figure represent questions, concepts, and relations, respectively. A dotted line indicates that a concept is mentioned in the title of a question. It can be seen that Q1 and Q2 use different terms for the need of sending email, *i.e.,* "SMTP message" and "email", "Yahoo", "Hotmail", "Gmail". The gap between the different expressions of Q1 and Q2 needs to be bridged by the background knowledge about the relations between related concepts such as "SMTP", "email", "Yahoo", "Hotmail", "Gmail". These relations may be domain specific and hard to be captured by word embedding models such as Word2Vec or BERT. Moreover, the user often requires explicit explanations about the relations to understand, for example,

M. Liu, S. Yu, X. Peng, X. Du, T. Yang, H. Xu, and G. Zhang are with the School of Computer Science and Shanghai Key Laboratory of Data Science, Fudan University, China.

X. Peng is the corresponding author (pengxin@fudan.edu.cn).

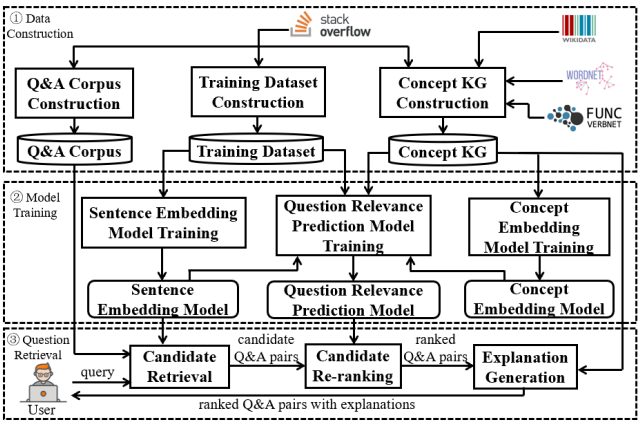

Fig. 2. An Overview of KGXQR

how SMTP works and what alternative protocols can be used for sending an email.

Based on the analysis, we propose KGXQR, a knowledge graph based question retrieval approach for software programming tasks. KGXQR employs BERT-based sentence similarity to retrieve relevant SO questions for a given query. To bridge the knowledge gap and enhance the performance of question retrieval, it constructs a software development related concept knowledge graph and trains a question relevance prediction model to re-rank the candidate questions. The model is trained based on a combined sentence representation of BERT-based sentence embedding and graph-based concept embedding. To help understand the relevance of the returned SO questions, KGXQR further generates explanations based on the association paths between the concepts involved in the query and the SO questions.

We constructed a knowledge graph comprising 2,291,354 concepts and 4,220,946 relations for KGXQR. We evaluate KGXQR using a benchmark consisting of three datasets, including test data from a previous study, SO duplicate question records, and SO title edit records. Experimental results demonstrated that KGXQR surpassed two baselines in terms of accuracy, recall, mean reciprocal rank (MRR), and mean average precision (MAP). Moreover, a user study with 10 participants and 16 programming tasks showed that the explanations provided by KGXQR can help them find answers to their questions faster and more accurately.

To summarize, this paper makes the following contributions:

- Creation of a training dataset and a benchmark consisting three test datasets for relevant question retrieval.
- Construction of a concept knowledge graph for software development with 2,291,354 concepts and 4,220,946 relations.
- Proposal of KGXQR, a knowledge graph based explainable question retrieval approach for software development that uses a retrieval and re-rerank pipeline.

II. Approach

As shown in Figure 2, KGXQR includes three phases, *i.e.,* data construction, model training, and question retrieval.

Data Construction. In this phase, KGXQR constructs a corpus of SO questions and answers (Q&A corpus), a training dataset, and a knowledge graph. The Q&A corpus includes high-quality Q&A pairs extracted from SO and provides the basis for question retrieval. The training dataset includes pairs of relevant and irrelevant question titles that are extracted from the post edit records and post linking records of SO. The dataset will be used to train the BERT-based sentence embedding model and the question relevance prediction model. The knowledge graph includes software development related concepts and their relations extracted from SO posts and comments and general knowledge graphs (*e.g.,* Wikidata [16]). It also includes conceptual-semantic and lexical relations enriched from lexical databases (*e.g.,* WordNet [17]) and categories of functionality verbs (*e.g.,* FuncVerbNet [18], [19]).

Model Training. In this phase, KGXQR trains a BERT-based sentence embedding model, a graph-based concept embedding model, and a question relevance prediction model. The sentence embedding model is trained by fine-tuning the pre-trained BERT model using contrastive learning based on the training dataset consisting of pairs of relevant questions. The model can map sentences into a high-dimensional dense vector space where relevant sentences are close. The concept embedding model is trained based on the knowledge graph. The model can map the concepts and relations of the knowledge graph to high-dimensional dense vectors which encode the structure information of the knowledge graph. The question relevance prediction model is trained to predict the relevance of two questions based on the training dataset. The model is defined based on a combined sentence representation of BERT-based sentence embedding and graph-based concept embedding.

Question Retrieval. In this phase, KGXQR returns a ranked list of Q&A pairs along with explanations for their relevance to the user query. It first uses BERT-based sentence embeddings to select candidate Q&A pairs based on similarity to the query. Next, the question relevance prediction model is used to re-rank the candidate pairs based on their predicted relevance. Finally, an explanation is generated for each returned Q&A pair based on the association paths between the concepts in the query and the SO question.

A. Data Construction

This phase includes three steps that construct the Q&A corpus, the training dataset, and the concept KG, respectively.

1) Q&A Corpus Construction: We extract high-quality Q&A pairs from SO to create a Q&A corpus for the four most popular programming languages: Java, Python, JavaScript, and C#. For each programming language, we select the questions that meet all the following three conditions: 1) the question is tagged with the corresponding programming language (*e.g.,* "java"); 2) the score of the question is greater than or equal to 5; 3) the question has an accepted answer whose score is greater than or equal to 5. For each selected question, we create a Q&A pair consisting of the question and its accepted answers. Using the SO data dump from September 2021 [2], we obtain

86,507, 81,064, 79,082, and 87,002 Q&A pairs for Java, Python, JavaScript, and C#, respectively.

2) Training Dataset Construction: The training dataset consists of pairs of relevant question titles (positive samples). To avoid manual data annotation, we automatically construct the dataset based on the duplicate question records and the question title edit records of SO.

SO marks duplicate questions which can be used as a data source of the training dataset. For example, *"How is Hashtable different to Hashmap"* [20] and *"What are the differences between a HashMap and a Hashtable in Java?"* [21] are marked as duplicate questions and we can treat their titles as a pair of relevant question titles. However, some duplicate questions may have low-quality titles that express irrelevant intentions. For example, *"ArrayList <? super Number> and Double"* [22] and *"What is PECS (Producer Extends Consumer Super)?"* [23] are duplicate questions, but we cannot learn relevant intentions from the two titles. To ensure the quality of the dataset, we only select duplicate question pairs that meet the following two conditions as the positive samples: 1) the two questions both have a positive score; 2) the Jaccard similarity [24] (a token-based lexical similarity) of their titles is greater than 0.1. We use a Python library TextDistance [25] to calculate the Jaccard similarity.

SO records the edit history of question titles. For example, the title of the SO question 345194 [26] was changed from "Regular expression matching in jQuery" to "Regular expression field validation in jQuery". The titles before and after the change express similar intentions and thus can be treated as a pair of relevant question titles. Similar to duplicate questions the edit history of a question may include low-quality titles that express irrelevant intentions. For example, the title *"How to access objects in EL expression language ${}"* [27] was changed from *"Help JSTL foreach"*, but we cannot learn relevant intentions from the two titles. To ensure the quality of the dataset, we only select at most one pair of titles that meet the following three conditions from the edit history of a question as a positive sample: 1) the question has a positive score; 2) the two titles are not the same after removing punctuations and spaces and converting to lowercase; 3) the Jaccard similarity of the two titles is greater than 0.1.

The SO dump used in our implementation includes 1,096,708 duplicate question records and 2,554,062 edit records for 16,663,358 questions. As a result, we sample 21,172 and 80,000 positive samples from the duplicate question records and the question edit records respectively.

3) Concept KG Construction: The software development related concept knowledge graph is constructed by extracting and combining concepts and relations extracted from different sources, including SO discussions and tags, general knowledge graphs, and lexical databases.

1. Knowledge Extraction from SO Tags and Discussions

SO tags are usually software development related concepts such as programming languages (*e.g.,* "java") and software techniques (*e.g.,* "md5"). For each tag SO collects a set of synonyms of it, for example "c sharp" is a synonym of "c#". We add all the SO tags and their synonyms into the knowledge graph and create synonymy relations between them. Besides tags, there are also many software development related concepts mentioned in SO posts (including questions and answers) and comments and most of them are domain-specific ones that are rarely found elsewhere, *e.g.,* "StringBuffer". We use an existing tool AutoPhrase [28] to extract such concepts from the text corpus of SO. AutoPhrase is an automated tool that can mine high-quality phrases from massive text corpus through a learning-based method. It does not rely on manual annotations, but uses the titles of Wikipedia pages as positive samples. Moreover, it does not require us to provide negative samples. To better reflect the characteristics of software development related concepts, we add all the SO tags and their synonyms into the set of positive samples. To prepare the text corpus to be mined, we collect all the SO posts and comments from the data dump. In addition, we implement a Scrapy [29] based crawler to obtain the description pages of all the SO tags (*e.g.,* "java" [30]) and add them into the text corpus. We then clean the text corpus by removing HTML tags and code snippets using BeautifulSoup [31]. Based on the text corpus and positive samples, AutoPhrase learns a set of models that can be used to identify high-quality phrases from the text corpus. We collect all the identified phrases with a confidence score higher than 0.75 and add them into the knowledge graph after lemmatization. Finally, we add the following two kinds of categorization relations between two concepts C_1 and C_2: 1) add a relation <C_2, facet of, C_1> if C_1's name is the prefix of C_2's name , *e.g.,* <email system, facet of, email>; 2) add a relation <C_2, is, C_1> if C_1's name is the suffix of C_2's name , *e.g.,* <email system, is, system>.

2. Knowledge Extraction from General Knowledge Graph

Wikidata [16] is a general knowledge graph with more than 97 million concepts. It includes concepts and relations that describe software development knowledge or related background knowledge, such as the concepts "Simple Mail Transfer Protocol" and "email system" and the relation <Simple Mail Transfer Protocol, alias, SMTP>. A challenge here is how to identify software development related concepts from the huge number of concepts of Wikidata. Therefore, we train a BERT-based binary text classifier to select software development-related concepts from Wikidata. The classifier takes the description of a Wikidata concept (*i.e.,* the corresponding Wikipedia article) as input and predicts whether the concept is related to software development or not. Some Wikidata concepts have corresponding SO tags (indicated by their "Stack Exchange tag" attributes) and these concepts can be regarded as software development related. We take all these Wikidata concepts as positive samples and randomly select twice the number of Wikidata concepts as negative samples considering that most Wikidata concepts are not related to software development. We obtain Wikidata concepts from the dump of Wikidata [32] and the corresponding Wikipedia articles from the dump of Wikipedia [33]. We divide the dataset into a training set and a validation set by 9:1 and the accuracy of the classifier is 87% in the validation set, indicating a high

accuracy. We use the trained classifier to identify software development-related concepts from Wikidata and add all the identified concepts together with their one-hop neighbors and relations into the knowledge graph. When adding the concepts to the knowledge graph, we merge them with the concepts from SO that have the same names.

3. Knowledge Extraction from Lexical Database

WordNet [17] is a large lexical database of English, which provides rich conceptual-semantic and lexical relations (*e.g.,* hyponymy, meronymy, entailment, synonym, antonym) for words (*i.e.,* nouns, verbs, adjectives, and adverbs). As some words in WordNet have the corresponding concepts in the concept knowledge graph, the relations in WordNet can be added to enrich the software development related knowledge, for example `<tag, hyponymy, label>`, `<chip, meronym, computer>`, `<authenticate, entailment, verify>`, and `<software, antonym, hardware>`. Therefore, we check each relation in WordNet and add it to the concept knowledge graph if the two words involved in the relation are both concepts in the knowledge graph. Verbs play an important role in the descriptions of software functionalities [18]. As a functionality often can be described using different verbs, the functionality categories and the corresponding common verbs are also important software development knowledge. For example, "convert" and "transform" belong to the same functionality category that converts something from a source to a target. Xie *et al.* [18] summarize 87 software functionality categories and the corresponding verbs, which are shared in an open-source project FuncVerbNet [19]. We add all the functionality categories and the verbs into the knowledge graph and create two kinds of relations between them: a *"belong to"* relation between a verb and the functionality category that the verb belong to; a *"same functionality category"* relation between two verbs that belong to the same functionality category. Note that a verb may belong to multiple functionality categories, as it may express different functionalities in different contexts.

Resulting Concept KG. The constructed background knowledge graph contains 2,291,354 software development-related concepts and 4,220,946 relations. Among them, 1,591,093 concepts and 1,883,557 relations are from Wikidata.

B. Model Training

In this phase, KGXQR trains a BERT-based sentence embedding model, a graph-based concept embedding model, and a question relevance prediction model.

1) BERT-based Sentence Embedding Model Training: A pre-trained BERT model can be directly used for sentence embedding. However, it may fail to capture domain-specific semantics without fine-tuning for specific tasks [34]. Therefore, we train a BERT-based sentence embedding model by fine-tuning the pre-trained BERT model using contrastive learning. The objective is to make the sentence vectors of relevant questions as close as possible and the those of irrelevant questions as far as possible.

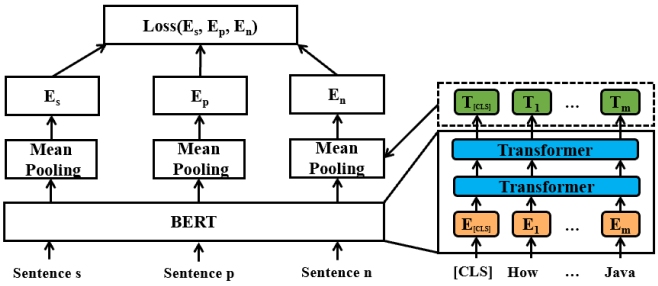

Fig. 3. Contrastive Learning for Fine-Tuning BERT Model

The architecture of the contrastive learning model is a triplet network [35] with two layers as shown in Figure 3. It takes three sentences s, p, and n as input. Among them, s is a base sentence, p is a positive sentence (i.e., relevant sentence) for s and n is a negative sentence (i.e., irrelevant sentence) for s. The first layer includes three identical deep neural network models (*i.e.,* BERT model) which respectively generate three sentence vectors E_s, E_p , and E_n for the input sentence s, p, and n by: 1) obtaining a contextual word vector of each word in the sentence using the BERT model; 2) calculating the sentence vector by averaging the word vectors of the sentence through an additional mean pooling layer. The BERT model will add a special token *[CLS]* at the beginning of the input sentence. The three BERT models share the same weights: they initialize the weights based on the same pre-trained BERT model and update the weights simultaneously while training. The second layer is a loss function calculated based on the distance between E_s, E_p, and E_n (see Equation 1), where $d(E_1, E_2)$ means the Euclidean distance between two vectors E_1 and E_2, σ is the margin of the two distances.

$$Loss(E_s, E_p, E_n) = max(d(E_s, E_p) - d(E_s, E_n) + \sigma, 0) \quad (1)$$

In the training process, the weights of the BERT models are simultaneously updated with the objective of minimizing the loss function. The objective implies to minimize the distance between the sentence vectors of s and p and maximize the distance between the sentence vectors of s and n.

The training dataset consists of pairs of relevant question titles and pairs of irrelevant question titles. The pairs of relevant question titles are from the training dataset introduced in Section II-A2. For each relevant pair <q_1, q_2> we identify an irrelevant pair <q_1, q_3> that shares a question title with it (*i.e.,* q_1) by randomly sampling q_3 from all questions in the corpus, and then use q_1, q_2, and q_3 as the base sentence, positive sentence, and negative sentence, respectively.

As for the pre-trained BERT model, we use the seBERT [36] model [37] instead of the official pre-trained BERT model [38]. seBERT is trained on a large software engineering corpus consisting of SO posts and GitHub commit messages, thus is more suitable for the tasks in the software engineering domain (*e.g.,* issue type classification) [36]. We fine-tune the seBERT model with sentence-transformers [39], a Python library for training sentence embedding model.

2) Concept Embedding Model Training: The knowledge (*i.e.,* concepts and relations) contained in the concept knowledge

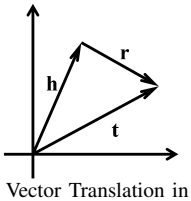

Fig. 4. Vector Translation in TransE

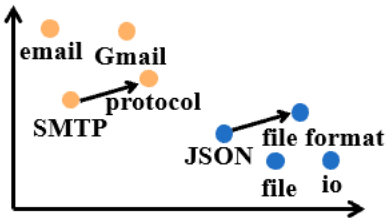

Fig. 5. Examples of Concept Embeddings using TransE

graph can help enhance the performance of question retrieval. To incorporate the knowledge into the question relevance prediction model we need to train a concept embedding model to map the concepts and relations in the knowledge graph to high-dimensional dense vectors encoding the structure information of the knowledge graph.

We use the TransE model [40], an energy-based graph embedding model, to train the concept embedding model. It takes all the relation triples in the knowledge graph as input and outputs the embeddings of all the concepts and relations. As shown in Figure 4 TransE models relations as translations operating on the low-dimensional embeddings of the entities (*i.e.,* concepts in KGXQR) [40]. For a relation triple <h, r, t>, where h, r, and t respectively are the head entity, relation type, and tail entity, TransE produces the corresponding vector representations E_h, E_r, E_t that satisfy $E_h + E_r \approx E_t$. In the training process, TransE optimizes the embeddings for each relation triple <h, r, t> with the objective of minimizing $d(E_h + E_r, E_t) - d(E'_h + E_r, E'_t)$, where $d(E_1, E_2)$ is the distance between two vectors E_1 and E_2 and h' and t' are randomly selected negative sample pairs for the relation r.

Figure 5 shows some examples of concept embeddings using TransE. It can be seen that the vector representations of the concepts that are close in the knowledge graph (*i.e.,* concepts in the same color) are relatively close in the embedding space (*e.g.,* "Gmail" and "mail", "file" and "io"). Moreover, the pairs of concepts having the same relation type (*e.g.,* <SMTP, is, protocol> and <json, is, file format>) share a similar pattern in the relative relationships of their vector representations in the space.

Considering that our knowledge graph is large (including more than 2 million concepts and more than 4 million relations), we use PyTorch-BigGraph (PBG) [41] and its implementation shared on GitHub [42] to train the TransE model. PyTorch-BigGraph is a distributed system implemented by Facebook with the purpose of supporting the training of graph embedding models on large graphs.

3) Question Relevance Prediction Model Training: Figure 6 shows the architecture of the question relevance prediction model, which takes two questions (*i.e.,* p and q) as input and outputs a prediction of their relevance between 0 and 1. The model first produces a joint vector representation for the two questions and then classifies the two questions into relevant or not using a Softmax classifier. To obtain the joint vector representation, the model first generates vector representations for the two questions using BERT-based sentence embedding and concept-based sentence embedding respectively, and then concatenate their vector representations.

BERT-based Sentence Embedding. The BERT-based sentence embeddings of p and q (*i.e.,* $V_{BERT,p}$ and $V_{BERT,q}$) are generated using the BERT-based sentence embedding model, as shown on the left side of Figure 6.

Concept-based Sentence Embedding. To generate the concept-based embedding (*i.e.,* V_{KG}) for a question (*i.e.,* p or q), we identify the concepts mentioned in the question and then generate a vector representation for the question based on the concept embeddings of its concepts. The two steps are detailed below.

1) Concept Identification. This step identifies the concepts mentioned in a question (*i.e.,* p or q). We first extract all the n-gram phrases from the questions and treat them as possible concepts mentions after lemmatization. In our implementation we set n to 4 following existing practice in phrase identification [43]. Then we try to link each candidate concept mention to a concept in the knowledge graph based name matching. If a linked concept mention is a part of another one, we only keep the longer one.

2) IDF Weighted Pooling. This step performs IDF weighted pooling based on the embeddings of the concepts identified from the question (*i.e.,* p or q). In particular, we calculate a weighted average of the concept embeddings and use it as the concept-based embedding of the question. The weight of a concept measures the importance of the concept and is defined as the sum of the IDFs (Inverse Document Frequencies) [9] of all the words contained in the concept. The IDF model used in our implementation is trained on the Q&A corpus (see Section II-A1) using Gensim [44].

Feature Concatenation. We concatenate the BERT-based embeddings (*i.e.,* $V_{BERT,p}$, $V_{BERT,q}$) of the two questions and the concept-based embeddings (*i.e.,* $V_{KG,p}$, $V_{KG,q}$) of the two questions in the same way as in [45], [46]. First, we concatenate the BERT-based and concept-based sentence embeddings (*i.e.,* V_{BERT}, V_{KG}) of a question (*i.e.,* p or q) to obtain the combined sentence embedding of the question (*i.e.,* E_p, E_q). Then, we further concatenate E_p and E_q in the following three ways:

- (E_p, E_q), *i.e.,* the concatenation of E_p and E_q;
- $|E_p - E_q|$, *i.e.,* the absolute value of element-wise difference between E_p and E_q;
- $|E_p * E_q|$, *i.e.,* the element-wise product between E_p and E_q.

We try all these different ways of concatenation and find that combining all of them achieves the best performance on the test set. Therefore, we concatenate (E_p, E_q), $|E_p - E_q|$, $|E_p * E_q|$ together to form the input of the Softmax classifier.

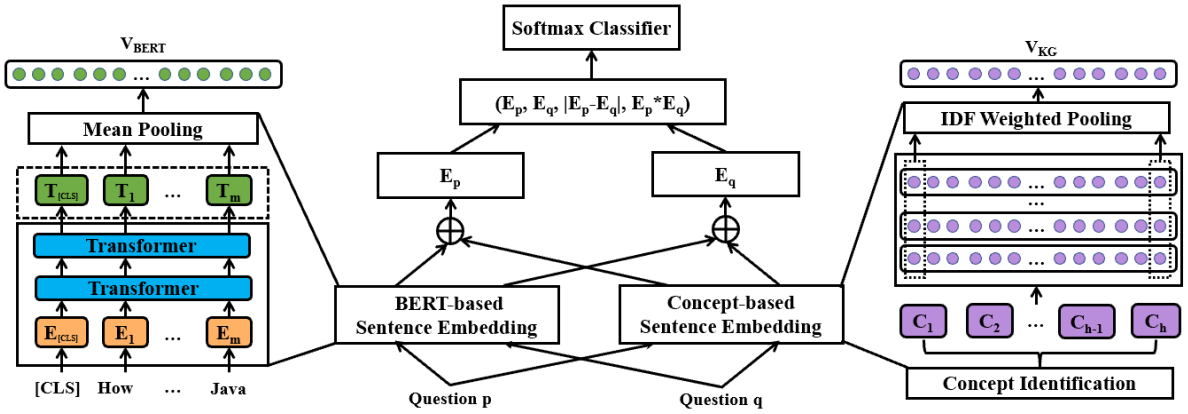

Fig. 6. Architecture of Question Relevance Prediction Model

We implement the question relevance prediction model with PyTorch 1.10.1. We train the model using a training set with a 1:8 ratio of positive and negative samples. This distribution reflects the nature of the Q&A corpus where relevant questions are much less than irrelevant questions. The positive samples are all from the dataset introduced in Section II-A2. We create eight negative samples for each positive sample by finding irrelevant questions for a question q in the positive sample using the following two strategies:

1) randomly sample four questions whose ranks range from 100 to 150 in the relevant questions of q based on the BERT-based sentence embedding model (see Section II-C1);

2) randomly sample four questions from the top-50 relevant questions of q produced by the retrieval module of AnswerBot [47], which are not included in the top-50 relevant questions produced by the BERT-based sentence embedding model at the same time.

We avoid selecting questions that are included in the positive samples involving q to create high-quality negative samples. These negative samples are considered hard negatives, and their distance in the embedding space is not significantly far from the positive samples. This sampling strategy is superior to random sampling and helps improve the quality of the generated negative samples [48].

C. Question Retrieval

Given a user query, KGXQR returns a list of ranked Q&A pair from the Q&A corpus, together with explanations for their relevance to the query. This phase includes three steps, *i.e.,* candidate retrieval, candidate re-ranking, and explanation generation.

1) Candidate Retrieval: The question relevance prediction model combines BERT-based sentence embedding and concept-based sentence embedding, thus can better estimate the relevance between the query and the questions. However, it requires an independent prediction for each pair of query and question and is time consuming. Therefore, before using the question relevance prediction model we first use a sentence embedding based method, which is much more efficient, to identify a set of candidate Q&A pairs.

We retrieve candidate Q&A pairs from the Q&A corpus (see Section II-A1) based on the BERT-based sentence embedding model (see Section II-B1). First, we use the BERT-based sentence embedding model to generate vector representations for both the query and the titles of the questions in the Q&A corpus. Second, we calculate the cosine similarities between the vector representations of the query and the question titles. Third, we rank all the questions by the similarity and take the top-k (*e.g.,* top-20) questions as candidates.

To improve the efficiency of candidate retrieval, we calculate the vector representations of all the questions in the Q&A corpus in advance and store them in Milvus [49]. Milvus is an open-source vector database [50] which supports high-efficient vector similarity search. In this way, we only need to calculate the query vector and retrieve the top-k similar question vectors using Milvus when retrieving candidate questions online. When the Q&A corpus grows, we can incrementally calculate vector representations for new questions and add them to Milvus.

2) Candidate Re-ranking: We use the question relevance prediction model (see Section II-B3) to re-rank all the candidate questions. For each candidate question, we treat the query and question title as the sentence p and q respectively and combine them as the input to the model. The model outputs a prediction ranging from 0 to 1 indicating the relevance between the query and the question. Then we re-rank all candidate questions based on the relevance prediction and return the results to the user.

3) Explanation Generation: For each returned question, we generate explanations for its relevance to the query as shown in Table I. The explanations are generated based on the concept knowledge graph (see Section II-A3) and describe how the query and question are relevant conceptually. The explanations include the following four parts.

- **Common Concepts**: the concepts that are directly mentioned in both the query and the question.
- **Association Paths**: the conceptual association paths between the concepts mentioned in the query and the question.

TABLE I
AN EXAMPLE OF GENERATED EXPLANATIONS

Query	How do I **send** an **SMTP** Message from **Java**?
Question	How can I **send** an **email** by **Java** application using **GMail**, **Yahoo**, or **Hotmail**?
Explanation	*Common Concepts:* **send**, **Java**
	Association Paths: 1. **Java**<-programming language<-**Gmail**; 2. **SMTP**->alias->Simple Mail Transfer Protocol->part of ->**email**<-use<-email system<-instance of<-**Hotmail**; 3. **SMTP**->alias->Simple Mail Transfer Protocol->part of ->**email**<-use<-email system<-instance of<-**Gmail**.
	Question Summary: Is it possible to **send** an **email** from my **Java** application using a **GMail** account? ... Answers with any of using **Hotmail**, **Yahoo** or **GMail** are acceptable
	Answer Summary: ...Here's a full working example using **GMail**

- **Question Summary**: the sentences excerpted from the question body that mention related concepts.
- **Answer Summary**: the sentences excerpted from the accepted answer that mention related concepts.

Related concepts refer to the concepts in the query, question, and association paths.

For a given query q and a returned question q', we first identify the set of concepts $C(q)$ and $C(q')$ mentioned in q and q' respectively following the same way in question relevance prediction model training (see Section II-B3). The common concepts thus can be identified as the intersection of $C(q)$ and $C(q')$. Then for each concept c in $C(q)$, we identify candidate paths from c to any concept in $C(q')$ in the concept knowledge graph. For efficiency, we limit the maximum length of the path to 4. For each candidate path p, we further estimate its quality considering its length and the relevance to the query q and the question q' using Equation 2, where $len(p)$ is the length of p, V_p is the vector representation of p, and $cos(V_1, V_2)$ is the cosine similarity of two vectors. The vector representations of p (*i.e.,* V_p) and q/q' (*i.e.,* $V_{q,q'}$) are generated by averaging the vector representations of all the words in their descriptions using a pre-trained Word2Vec model. The description of the path p consists of the names of all the concepts and relations involved in the path. The description of q and q' consists of the query (*i.e.,* q) and the question (*i.e.,* q') themselves. The Word2Vec model is trained on the Q&A corpus (see Section II-A1) with Gensim [44]. We rank all the candidate paths by the quality estimation and select the top-5 paths as the association paths.

$$Quality(p) = 1/len(p) * (cos(V_p, V_{q,q'}) + 1)/2 \quad (2)$$

To generate the question summary, we split the question body into sentences, rank the sentences by the number of related concepts, and select the top-2 sentences as the summary. Note that sentences without any related concepts are filtered out. If two sentences have the same number of related concepts, we prefer the sentence that contains more related concepts that are not covered by other selected sentences to ensure diversity. The answer summary is generated in the same way.

TABLE II
ACCURACY OF CONCEPTS AND RELATIONS

Type	Concepts/SO	Relations/SO	Concepts/Wikidata
Accuracy	89.5%	78.1%	93.2%
Agreement	0.87	0.80	0.72

III. EVALUATION

We conduct a series of experiments to evaluate the intrinsic quality of the concept knowledge graph and the effectiveness and usefulness of KGXQR by answering the following research questions. All the data and results can be found in our replication package [51].

RQ1 (KG Intrinsic Quality): What is the intrinsic quality of the concept knowledge graph constructed for KGXQR?

RQ2 (Effectiveness): How well does KGXQR perform in relevant question retrieval compared with the baselines?

RQ3 (Explainability): Are the explanations provided by KGXQR helpful for the user in relevant question retrieval?

A. RQ1: KG Intrinsic Quality

We evaluate the intrinsic quality of the concept knowledge graph by assessing the correctness of the concepts and relations in the knowledge graph.

1) Protocol: To ensure that our observations generalize to the population within a certain confidence interval at a certain confidence level, we use a sampling method, as done in previous studies [52], [53], [54]. A sample size of 384 is required for a confidence interval of 5 at a 95% confidence level. As described in Section II-A3, we construct the knowledge graph by extracting knowledge from SO tags and discussions, the general knowledge graph (*i.e.,* Wikidata), and lexical databases. We randomly sample 384 concepts and 384 relations extracted from SO tags and discussions, and 384 concepts extracted from Wikidata. Relations extracted from Wikidata and lexical databases are not evaluated since they are already manually verified in their sources. We invite four MS students majored in software engineering to independently annotate the sampled concepts and relations. For each concept, two students decide whether it is related to software development or not. For each relation, two students annotate it to be correct or not. If their annotations differ, a third student is assigned to give an additional annotation to resolve the conflict by a majority-win strategy.

2) Results: Table II shows the accuracy of sampled concepts and relations with their Cohen's Kappa agreements [55]. The Kappa agreements are all above 0.7, indicating substantial or almost perfect agreement. The concepts extracted from SO (*i.e.,* tags and discussions) and Wikidata have generally higher accuracy, 89.5% and 93.2%, respectively. The accuracy of the relations extracted from SO is slightly lower (78.1%), but still acceptable. The reason for the low accuracy of relations is that they are extracted using simple heuristic rules based on names. In future work, more advanced techniques (*e.g.,* deep learning-based relation extraction techniques [56]) will be applied to extract more accurate relations from SO. Moreover, we can identify low-quality concepts and relations in the knowledge

graph through crowdsourcing or using deep learning techniques. We will make our concept knowledge graph open for the research community and keep it updated and maintained.

3) Summary: The concept knowledge graph is of high quality, which is indicated by the accuracy of 91.4% for the sampled concepts and 78.1% for the sampled relations.

B. RQ2: Effectiveness

We evaluate the effectiveness of KGXQR for relevant question retrieval by comparing it with baselines on different test datasets.

1) Baselines: We compare KGXQR with baselines from previous work, *i.e.,* AnswerBot-Ret [6] and CLEAR-Ret [57].

AnswerBot-Ret. AnswerBot [6] includes a module for relevant question retrieval, which combines the Word2Vec model and IDF metric to measure the relevance between the input query and the questions in the corpus. We call this module Answerbot-Ret and obtain its implementation from the replication package of AnswerBot [58]. This baseline is a representation of Word2Vec-based methods.

CLEAR-Ret. CLEAR is an API recommendation method based on crowd-sourced knowledge from Stack Overflow. It relies on a BERT-based question retrieval module to retrieve relevant SO questions for a given query, and then extracts candidate APIs from these related SO questions. We refer to their method as CLEAR-Ret. Similar to our approach for question retrieval (Section II-C1), CLEAR-Ret also trains a BERT-based sentence embedding model on a dataset composed of titles of relevant and irrelevant SO questions. Their dataset construction criterion is whether two SO questions' answers involve the same API. We adapt CLEAR-Ret as a baseline by training the BERT-based sentence embedding model on the same dataset of our approach, using the pre-trained BERT model [38].

We do not compare with traditional IR methods such as TF-IDF, as AnswerBot has shown its advantage over them [6].

2) Test Datasets: We create a benchmark consisting of three test datasets as shown in Table III. The three datasets have different characteristics. The AnswerBot dataset is human annotated small datasets and contain only Java-related queries, while duplicate question dataset and title edit dataset are automatically created large datasets and cover the four most popular programming languages. The details of these datasets are as follows:

AnswerBot Dataset. We used the AnswerBot replication package [58] which provides a dataset of 100 Java-related queries and their relevant SO questions as ground truth. The queries are randomly selected SO question titles and the ground truth includes manually annotated relevant SO questions from the top-10 retrieval results produced by AnswerBot-Ret and other baselines. However, as our Q&A corpus is different from theirs, some relevant questions for a query might not be included in their dataset. Therefore, we invite two MS students to identify more relevant questions for the quires following the same way in [58]. For each query we combine the top-10 questions returned by KGXQR and all the baselines as the candidates. Then the two students annotate each candidate questions to determine whether it is relevant to the query or not. If their annotations are different, a third student is assigned to give an additional annotation to resolve the conflict by a majority-win strategy. The Kappa agreement [55] for the annotation is 0.662, *i.e.,* substantial agreement. In the resulting dataset the set of relevant questions of each query consists of the questions identified by the annotators and the questions in the original dataset.

TABLE III
OVERVIEW OF ALL TEST DATASETS

Test Dataset	Query Number	Ground Truth Number
AnswerBot Dataset	100	626
Duplicate Question Dataset	4,000	12,687
Title Edit Dataset	8,000	8,000

Duplicate Question Dataset. We randomly select 1,000 questions that have duplicates for four programming languages (*i.e.,* Java, Python, JavaScript, and C#) respectively from the SO data dump [2] as queries and all their corresponding duplicate questions as the ground truth. Note that a question may have multiple duplicate questions.

Title Edit Dataset. We randomly select 2,000 questions with title edit records for four programming languages (*i.e.,* Java, Python, JavaScript, and C#) respectively from the SO data dump [2]. For each question we use one of its historical titles as the query and its current title as the relevant question.

We ensure that all the datasets have no overlaps with our training dataset. Further, we add the ground truth (*i.e.,* relevant questions) of all the queries in these datasets into the Q&A corpus we constructed in Section II-A1.

3) Protocol: We evaluate KGXQR and the two baselines on the three test datasets based on the same Q&A corpus. KGXQR and CLEAR-Ret use the same training data and hyperparameters. In candidate retrieval KGXQR selects top-10 Q&A pairs as candidates for re-ranking. For AnswerBot we train a Word2Vec model for each of the four programming languages. As the queries are language specific, we retrieve relevant questions in the same language question for each query.

We use widely used IR metrics [59], *i.e.,* A@k, R@k, MRR (Mean Reciprocal Rank), and MAP (Mean Average Precision), for the evaluation. A@k indicates whether the top-k results contain at least one relevant question. R@k is the recall in the top-k results. MRR reflects the ranking of the first relevant question in the returned results. MAP reflects the rankings of all the relevant questions. All metrics we use are the higher the better.

4) Results: The evaluation results are shown in Table IV, where the best value of each metric is in boldface. Overall, KGXQR outperforms all the baselines on all the three datasets.

Both KGXQR and CLEAR-Ret outperform AnswerBot-Ret across all three datasets, indicating the advantage of BERT-based sentence embedding. However, KGXQR and CLEAR-Ret have lower A@1 and R@1 performance than AnswerBot-Ret on

TABLE IV
PERFORMANCE COMPARISON BETWEEN KGXQR AND BASELINES ON DIFFERENT TEST DATASETS

Dataset	Metric	AnswerBot-Ret	CLEAR-Ret	KGXQR
AnswerBot Dataset	A@1	0.290	0.590	**0.810**
	A@3	0.560	0.750	**0.910**
	A@5	0.620	0.800	**0.930**
	A@10	0.710	0.870	**0.950**
	R@1	0.052	0.120	**0.167**
	R@3	0.147	0.257	**0.361**
	R@5	0.177	0.318	**0.497**
	R@10	0.251	0.435	**0.678**
	MRR	0.435	0.683	**0.863**
	MAP	0.150	0.326	**0.546**
Duplicate Question Dataset	A@1	**0.092**	0.001	0.001
	A@3	0.166	0.238	**0.278**
	A@5	0.207	0.333	**0.393**
	A@10	0.271	0.453	**0.525**
	R@1	**0.069**	0.000	0.000
	R@3	0.122	0.174	**0.204**
	R@5	0.152	0.243	**0.292**
	R@10	0.199	0.333	**0.393**
	MRR	0.142	0.144	**0.167**
	MAP	0.106	0.107	**0.126**
Title Edit Dataset	A@1	0.663	0.739	**0.814**
	A@3	0.738	0.814	**0.863**
	A@5	0.153	0.762	**0.871**
	A@10	0.790	0.862	**0.875**
	R@1	0.663	0.739	**0.814**
	R@3	0.738	0.814	**0.863**
	R@5	0.762	0.835	**0.871**
	R@10	0.790	0.862	**0.875**
	MRR	0.706	0.781	**0.839**
	MAP	0.706	0.781	**0.839**

the duplicate question dataset. We attribute this to two reasons. Firstly, the ground truth in the duplicate question dataset is incomplete, and the annotated ground truth tends to come from questions with significant semantic differences in their titles. As a result, KGXQR and CLEAR-Ret may rank other questions that are semantically closer but not in the ground truth higher, which are also correct. Secondly, the BERT-based sentence embedding model primarily considers semantic-level similarity including the sentence pattern, which may result in slightly lower performance for correct answers that are more lexically similar to the query but share fewer keywords. This is where the Word2Vec-based method AnswerBot performs better. However, in the duplicate question dataset, KGXQR and CLEAR-Ret significantly outperform AnswerBot-Ret in A@3, A@5, R@3, and R@5 performance. One possible way to further improve the effectiveness of our approach is to combine it with a lexical similarity-based retrieval approach.

It is important to note that AnswerBot's outcomes differ from the results reported in the original paper, primarily due to the utilization of a distinct Q&A corpus for retrieval, with the former employing a more comprehensive and up-to-date one.

We analyze some of the testing data where our method does not perform well and identify typical reasons for errors. These include insufficient coverage or quality issues in the concept knowledge graph, as well as irrelevant concepts identified from the question title that lead to confusion, or a lack of useful concept mentions in the question title (with some mentioned only in the question body).

Our analysis suggests some possible ways to improve KGXQR. First, to better capture the conceptual associations between queries and questions, the concept knowledge graph could be expanded to include more high-quality concepts and relations, and noise reduction methods could be employed to remove low-quality or irrelevant concepts. Second, incorporating an attention layer [60] into the concept-based sentence embedding could reduce the impact of irrelevant concepts. Third, including the question body could provide more information for question retrieval, but noise reduction methods [3] should be employed to mitigate its effects.

5) Summary: The evaluation results demonstrate that KGXQR is effective in related question retrieval, as it outperforms the baselines on various evaluation metrics across all three datasets.

C. RQ3: Explainability

To evaluate the usefulness of the explanations provided by KGXQR, we compare the performance of users in selecting relevant questions for specific programming tasks with the explanations generated by KGXQR and a baseline method respectively.

1) Baseline: We use TextRank [61], a general-purpose method for automatic text summarization as the baseline. It uses a graph-based ranking model to generate a summary for a given document. For a returned question, we use TextRank to extract up to three sentences from the question body and the answer respectively and combine them together as the explanation. We use the implementation of TextRank provided by Gensim [44].

2) Tasks and Participants: We select 16 Java-related programming tasks from the AnswerBot dataset and the duplicate question dataset randomly. We ensure that the relevant questions for each task are within the top-10 retrieval results of AnswerBot-Ret. The 16 tasks are divided into two roughly equal task groups (TA and TB) based on the rankings of their relevant questions. We invite 10 MS students to participate in the study. Before the study, we conduct a pre-study survey to assess their Java programming experience and divide them into two roughly equal participant groups (PA and PB) based on the survey results.

3) Protocol: For each task, we ask the participants to select relevant questions which could provide help for the task from the top-10 retrieval results of AnswerBot-Ret. They complete the tasks with the help of the explanations generated by KGXQR or TextRank. In particular, the participants in PA complete TA with KGXQR and TB with TextRank; the participants in PB complete TA with TextRank and TA with KGXQR. The tasks are interleaved for each participant. That is, the participant completes one task with KGXQR and one with TextRank. For each task we provide the original SO question (including title and question body) as the context for the participants. The participants can submit nothing or multiple questions for the task. They have a time limit of 10 minutes for each task and are considered to submit nothing if

they cannot finish in time. We record the relevant questions they submit and their completion time.

After they finish all the tasks we conduct an survey to collect their feedback. They are asked to evaluate the explanations generated by KGXQR and TextRank in terms of readability and usefulness on a 4-points Likert scale [62] (1-disagree; 2-somewhat disagree; 3-somewhat agree; 4-agree) by the following statements:

- **Readability**. The explanations generated by KGXQR (TextRank) are well-organized and easy to understand.
- **Usefulness**. The explanations generated by KGXQR (TextRank) are useful for identifying relevant questions.

4) Results: For the 16 tasks, the participants submit 157 non-empty results (3 empty results with TextRank), which include 454 questions in total (240 with KGXQR and 214 with TextRank) . For a task, we consider its submitted result to be correct as long as it contains at least one relevant question from its ground truth. As a result, 111 submitted results (59 results with KGXQR and 52 results with TextRank) are assessed as correct and we exclude incorrect ones when analyzing the completion time.

Table V shows the accuracy and the completion time of the participants with the explanations generated by KGXQR and TextRank respectively. It can be seen that, when they are provided with explanations generated by KGXQR, the participants complete the tasks 13.5% more accurately and 4.5% faster. The improvement in time is not so significant. After analysis, we believe that the reasons lie in the following two aspects: 1) the explanations of KGXQR contain more information (*e.g.,* association paths), thus the participants need to spend more time reading; 2) the explanations of KGXQR help the participants notice some relevant questions that are ignored when using the explanations of TextRank, thus they spend more time confirming the results and submit more questions (240 questions vs 214 questions). We further analyze the average time to identify a relevant question. We find that on average using the explanations of KGXQR helps the participants to find a relevant question 20.7% faster than using TextRank (23s vs 29s).

Table VI shows the results of the readability and usefulness evaluation. Overall, the participants found the explanations generated by KGXQR to be more readable and useful compared to those generated by TextRank. Almost all participants preferred KGXQR's explanations, except for one. Participants found that KGXQR's explanations highlighted key concepts in the query and the returned results, and provided informative sentences around related concepts. This helped them better understand the relevance between the query and the questions and increased their confidence in the selected questions. The one participant who gave negative feedback mentioned that KGXQR's association paths were sometimes redundant and preferred simple summaries with only relevant sentences. This suggests that future improvements could focus on generating more accurate and relevant conceptual association paths based on a higher-quality knowledge graph.

TABLE V
ACCURACY AND COMPLETION TIME OF TASKS WITH EXPLANATIONS GENERATED BY KGXQR AND TEXTRANK RESPECTIVELY

	Avg. Accuracy	**Avg. Time**
TextRank	65.0%	177s
KGXQR	73.8%	169s

TABLE VI
READABILITY AND USEFULNESS OF EXPLANATIONS GENERATED BY KGXQR AND TEXTRANK RESPECTIVELY

Aspect	**Readability**				**Usefulness**			
	1	2	3	4	1	2	3	4
TextRank	0	2	4	4	0	0	4	6
KGXQR	0	0	5	5	0	0	1	9

5) Summary: The explanations provided by KGXQR led to a 13.5% increase in accuracy and a 20.7% reduction in time to find relevant questions, compared to the baseline. Participants also found the explanations to be readable and useful.

D. Threats to Validity

Our studies face two main threats to internal validity: subjective judgment in human annotations and potential quality issues or biases in the datasets. To address these threats, we use commonly used data analysis principles such as assigning multiple annotators, conflict resolution, and reporting agreement coefficients for intrinsic quality evaluation. We also minimize human interventions in dataset construction by constructing the training and test datasets automatically to avoid bias caused by manual annotation. Additionally, we ensure that the training and test datasets have no overlaps, and share all datasets used in our studies in our replication package [51] for further evolution, correction (if needed), and reuse by other researchers.

A limitation to the external validity of our studies is the limited number of subjects (*e.g.,* programming languages, Q&A pairs in the corpus, tasks, and participants) considered in the evaluation. To address this limitation, we evaluate the effectiveness of KGXQR on three datasets of different characteristics (*e.g.,* sizes, programming languages, construction methods) to demonstrate its generalization. However, our findings may not be applicable to broader software development questions in practice.

IV. RELATED WORK

Researchers have explored several analysis tasks for SO questions, including question classification [3], duplicate question detection [63], [64], and question quality prediction [65].

Some researchers use SO question retrieval as a critical step for other tasks such as answer summary generation [6] and API recommendation [8]. For example, Huang *et al.* [8] retrieve questions related to a programming task and extract candidate APIs from them. Some researchers focus on retrieving specific types of SO questions, such as API-related ones [3]. Traditional IR techniques like TF-IDF [9] and BM25 [10] are commonly used for question retrieval [4]. To address the lexical gap between queries and questions, Xu *et al.* [6] combine Word2Vec models and IDF metrics to measure relevance between queries and questions in the corpus.

Some researches retrieve specific parts of SO threads such as answers [66] and code snippets [7] using various inputs like APIs [67] and queries in other languages [68]. In contrast to these approaches, KGXQR leverages BERT-based sentence embedding and knowledge graph-based concept embedding to improve the performance of question retrieval by bridging the knowledge gap. Additionally, KGXQR generates explanations for relevant questions returned based on conceptual associations between the query and questions.

To improve the performance of retrieval tasks for software engineering tasks such as bug localization [69], code search [70], [71], and question retrieval [72], [73], [4]), some researchers propose approaches for automatically reformulating queries by supplementing relevant terms [74], [71], [69], [70], [75], [4], asking clarification questions [72], or incorporating query logs [73]. These approaches often rely on external knowledge to bridge the gap between the query and the documents. For example, Lu *et al.* [70] expand queries with synonyms based on WordNet for code search. Some other researchers have attempted to enhance relevance calculation by leveraging relationships between APIs [5], [76]. For instance, Lin *et al.* [5] developed an API graph that incorporates structural relations between APIs and computes relevance using weighted API graph embedding. Different from their work KGXQR constructs a conceptual knowledge graph and integrates concept-based embedding into the relevance prediction instead of directly using the graph for similarity calculation.

Researchers in software engineering have created knowledge graphs for various purposes such as API caveats [52], domain terminology [53], [77], [78], API concepts [76], [79], API comparison [54], API documentation [80], [81], programming tasks [82], ML/DL models [83], and bugs [84], [85]. However, the KGXQR project stands apart as it aims to construct a knowledge graph specific to software development concepts for the purpose of retrieving relevant Stack Overflow questions.

V. Conclusions and Future Work

This paper proposes KGXQR, a knowledge graph-based approach for explainable question retrieval in programming tasks. KGXQR builds a concept knowledge graph and trains a question relevance prediction model using a combination of BERT-based sentence embedding and graph-based concept embedding. The model re-ranks candidate SO questions selected based on BERT-based sentence similarity to enhance the performance of question retrieval. Additionally, it generates explanations based on the knowledge graph. The evaluation demonstrates the effectiveness of KGXQR for question retrieval and the usefulness of the generated explanations. Future endeavors will be directed towards augmenting and broadening the existing approach to facilitate knowledge-based comprehension and recommendation of Stack Overflow discussions. Additionally, investigations will be conducted to explore the integration of more advanced concept embedding technologies for further improvement.

Acknowledgment

This work is supported by National Natural Science Foundation of China under Grant No. 61972098.

References

[1] C. Treude, O. Barzilay, and M. D. Storey, "How do programmers ask and answer questions on the web?" in *33rd International Conference on Software Engineering, ICSE 2011, May 21-28, 2011, Waikiki, Honolulu, HI, USA*. ACM, 2011, pp. 804–807.

[2] (2021) Stack overflow data dump version from september 4, 2021. [Online]. Available: https://archive.org/download/stackexchange/

[3] M. Liu, X. Peng, A. Marcus, S. Xing, C. Treude, and C. Zhao, "Api-related developer information needs in stack overflow," *IEEE Transactions on Software Engineering*, 2021.

[4] M. Liu, X. Peng, Q. Jiang, A. Marcus, J. Yang, and W. Zhao, "Searching stackoverflow questions with multi-faceted categorization," in *Proceedings of the Tenth Asia-Pacific Symposium on Internetware*, 2018, pp. 1–10.

[5] Z. Lin, Y. Zou, J. Zhao, and B. Xie, "Improving software text retrieval using conceptual knowledge in source code," in *32nd IEEE/ACM International Conference on Automated Software Engineering, ASE 2017, Urbana, IL, USA, October 30 - November 03, 2017*. IEEE Computer Society, 2017, pp. 123–134.

[6] B. Xu, Z. Xing, X. Xia, and D. Lo, "Answerbot: Automated generation of answer summary to developersź technical questions," in *32nd IEEE/ACM International Conference on Automated Software Engineering, ASE 2017, October 30 - November 03, 2017, Urbana, IL, USA*, 2017, pp. 706–716.

[7] R. F. G. da Silva, C. K. Roy, M. M. Rahman, K. A. Schneider, K. Paixão, C. E. d. C. Dantas, and M. d. A. Maia, "Crokage: effective solution recommendation for programming tasks by leveraging crowd knowledge," *Empirical Software Engineering*, vol. 25, no. 6, pp. 4707–4758, 2020.

[8] Q. Huang, X. Xia, Z. Xing, D. Lo, and X. Wang, "API method recommendation without worrying about the task-api knowledge gap," in *33rd ACM/IEEE International Conference on Automated Software Engineering, ASE 2018, Montpellier, France, September 3-7, 2018*. ACM, 2018, pp. 293–304.

[9] G. Salton and C. Buckley, "Term-weighting approaches in automatic text retrieval," *Inf. Process. Manag.*, vol. 24, no. 5, pp. 513–523, 1988.

[10] S. E. Robertson and S. Walker, "Some simple effective approximations to the 2-poisson model for probabilistic weighted retrieval," in *Proceedings of the 17th Annual International ACM-SIGIR Conference on Research and Development in Information Retrieval. Dublin, Ireland, 3-6 July 1994 (Special Issue of the SIGIR Forum)*, W. B. Croft and C. J. van Rijsbergen, Eds. ACM/Springer, 1994, pp. 232–241.

[11] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Advances in Neural Information Processing Systems*, vol. 26. Curran Associates, Inc., 2013, pp. 3111–3119.

[12] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation," in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*, A. Moschitti, B. Pang, and W. Daelemans, Eds. ACL, 2014, pp. 1532–1543.

[13] J. Dev, M. Chang, K. Lee, and K. Toutanova, "BERT: pre-training of deep bidirectional transformers for language understanding," in *2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019 Volume 1 (Long and Short Papers), June 2-7, 2019, Minneapolis, MN, USA*, 2019, pp. 4171–4186.

[14] (2023) Stack overflow question 73580: How do i send an smtp message from java? [Online]. Available: https://stackoverflow.com/questions/73580

[15] (2023) Stack overflow question 46663: How can i send an email by java application using gmail, yahoo, or hotmail? [Online]. Available: https://stackoverflow.com/questions/46663

[16] (2022) Wikidata. [Online]. Available: https://wikidata.org/

[17] G. A. Miller, "Wordnet: A lexical database for english," *Commun. ACM*, vol. 38, no. 11, pp. 39–41, 1995.

[18] W. Xie, X. Peng, M. Liu, C. Treude, Z. Xing, X. Zhang, and W. Zhao, "API method recommendation via explicit matching of functionality verb phrases," in *28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/SIGSOFT FSE 2020, November 8-13, 2020, Virtual Event, USA*. ACM, 2020, pp. 1015–1026.

[19] (2022) Funcverbnet. [Online]. Available: https://github.com/FudanSELab/funcverbnet

[20] (2023) Stack overflow question 13318139: How is hashtable different to hashmap. [Online]. Available: https://stackoverflow.com/questions/1331813

[21] (2023) Stack overflow question 40471: What are the differences between a hashmap and a hashtable in java? [Online]. Available: https://stackoverflow.com/questions/40471

[22] (2023) Stack overflow question 8083204: Arraylist <? super number> and double. [Online]. Available: https://stackoverflow.com/questions/40471

[23] (2023) Stack overflow question 2723397: What is pecs (producer extends consumer super)? [Online]. Available: https://stackoverflow.com/questions/2723397

[24] S. Niwattanakul, J. Singthongchai, E. Naenudorn, and S. Wanapu, "Using of jaccard coefficient for keywords similarity," in *Proceedings of the international multiconference of engineers and computer scientists*, vol. 1, no. 6, 2013, pp. 380–384.

[25] (2023) Textdistance. [Online]. Available: https://github.com/life4/textdistance

[26] (2023) Stack overflow question 345194: Regular expression field validation in jquery. [Online]. Available: https://stackoverflow.com/questions/345194

[27] (2023) Stack overflow question 5387174: How to access objects in el expression language ${}. [Online]. Available: https://stackoverflow.com/questions/5387174

[28] J. Shang, J. Liu, M. Jiang, X. Ren, C. R. Voss, and J. Han, "Automated phrase mining from massive text corpora," *IEEE Transactions on Knowledge and Data Engineering*, vol. 30, no. 10, pp. 1825–1837, 2018.

[29] (2023) Scrapy. [Online]. Available: https://scrapy.org/

[30] (2023) Javascript tag. [Online]. Available: https://stackoverflow.com/tags/javascript/info

[31] (2023) Beautifulsoup. [Online]. Available: https://www.crummy.com/software/BeautifulSoup/bs4/doc/

[32] (2021) Wikidata data dump version from november 24, 2021. [Online]. Available: https://dumps.wikimedia.org/wikidatawiki/entities/

[33] (2021) Wikipedia data dump version from december 20, 2021. [Online]. Available: https://dumps.wikimedia.org/enwiki/

[34] N. Reimers and I. Gurevych, "Sentence-bert: Sentence embeddings using siamese bert-networks," in *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, EMNLP-IJCNLP 2019, Hong Kong, China, November 3-7, 2019*. Association for Computational Linguistics, 2019, pp. 3980–3990.

[35] E. Hoffer and N. Ailon, "Deep metric learning using triplet network," in *Similarity-Based Pattern Recognition - Third International Workshop, SIMBAD 2015, Copenhagen, Denmark, October 12-14, 2015, Proceedings*, ser. Lecture Notes in Computer Science, A. Feragen, M. Pelillo, and M. Loog, Eds., vol. 9370. Springer, 2015, pp. 84–92.

[36] J. von der Mosel, A. Trautsch, and S. Herbold, "On the validity of pre-trained transformers for natural language processing in the software engineering domain," *CoRR*, vol. abs/2109.04738, 2021. [Online]. Available: https://arxiv.org/abs/2109.04738

[37] (2023) pre-trained sebert model. [Online]. Available: https://github.com/smartshark/seBERT

[38] (2023) pre-trained bert model. [Online]. Available: https://github.com/google-research/bert

[39] (2022) sentence-transformers. [Online]. Available: https://github.com/UKPLab/sentence-transformers

[40] A. Bordes, N. Usunier, A. García-Durán, J. Weston, and O. Yakhnenko, "Translating embeddings for modeling multi-relational data," in *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States*, 2013, pp. 2787–2795.

[41] A. Lerer, L. Wu, J. Shen, T. Lacroix, L. Wehrstedt, A. Bose, and A. Peysakhovich, "PyTorch-BigGraph: A Large-scale Graph Embedding System," in *Proceedings of the 2nd SysML Conference*, Palo Alto, CA, USA, 2019.

[42] (2023) Pytorch-biggraph. [Online]. Available: https://github.com/facebookresearch/PyTorch-BigGraph

[43] A. Handler, M. Denny, H. M. Wallach, and B. O'Connor, "Bag of what? simple noun phrase extraction for text analysis," in *1st Workshop on NLP and Computational Social Science, NLP+CSS@EMNLP 2016, November 5, 2016, Austin, TX, USA*. Association for Computational Linguistics, 2016, pp. 114–124.

[44] (2022) gensim. [Online]. Available: https://radimrehurek.com/gensim/

[45] M. Ostendorff, T. Ruas, M. Schubotz, G. Rehm, and B. Gipp, "Pairwise multi-class document classification for semantic relations between wikipedia articles," in *the ACM/IEEE Joint Conference on Digital Libraries in 2020, JCDL 2020, Virtual Event, China, August 1-5, 2020*. ACM, 2020, pp. 127–136.

[46] Y. Wang, Y. Yang, Y. Chen, J. Bai, C. Zhang, G. Su, X. Kou, Y. Tong, M. Yang, and L. Zhou, "Textnas: A neural architecture search space tailored for text representation," in *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*. AAAI Press, 2020, pp. 9242–9249.

[47] S. Gottipati, D. Lo, and J. Jiang, "Finding relevant answers in software forums," in *26th IEEE/ACM International Conference on Automated Software Engineering (ASE 2011), November 6-10, 2011, Lawrence, KS, USA*. IEEE Computer Society, 2011, pp. 323–332.

[48] J. Huang, A. Sharma, S. Sun, L. Xia, D. Zhang, P. Pronin, J. Padmanabhan, G. Ottaviano, and L. Yang, "Embedding-based retrieval in facebook search," in *26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, KDD 2020, Virtual Event, CA, USA, August 23-27, 2020*. ACM, 2020, pp. 2553–2561.

[49] J. Wang, X. Yi, R. Guo, H. Jin, P. Xu, S. Li, X. Wang, X. Guo, C. Li, X. Xu *et al.*, "Milvus: A purpose-built vector data management system," in *Proceedings of the 2021 International Conference on Management of Data*, 2021, pp. 2614–2627.

[50] (2023) milvus. [Online]. Available: https://github.com/milvus-io/milvus

[51] (2022) Replication package. [Online]. Available: https://kgxqr.github.io/

[52] 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 *34th IEEE International Conference on Software Maintenance and Evolution, ICSME 2018, September 23-29, 2018, Madrid, Spain*, 2018, pp. 183–193.

[53] 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 *27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/SIGSOFT FSE 2019, August 26-30, 2019, Tallinn, Estonia*. ACM, 2019, pp. 97–108.

[54] Y. Liu, M. Liu, X. Peng, C. Treude, Z. Xing, and X. Zhang, "Generating concept based API element comparison using a knowledge graph," in *35th IEEE/ACM International Conference on Automated Software Engineering, ASE 2020, September 21-25, 2020, Melbourne, Australia*. IEEE, 2020, pp. 834–845.

[55] M. L. McHugh, "Interrater reliability: the kappa statistic," *Biochemia Medica: Biochemia Medica*, vol. 22, no. 3, pp. 276–282, 2012.

[56] S. Kumar, "A survey of deep learning methods for relation extraction," *arXiv preprint arXiv:1705.03645*, 2017.

[57] M. Wei, N. S. Harzevili, Y. Huang, J. Wang, and S. Wang, "CLEAR: contrastive learning for API recommendation," in *44th IEEE/ACM 44th International Conference on Software Engineering, ICSE 2022, Pittsburgh, PA, USA, May 25-27, 2022*. ACM, 2022, pp. 376–387. [Online]. Available: https://doi.org/10.1145/3510003.3510159

[58] (2023) Answerbot replication package. [Online]. Available: https://github.com/maxxbw54/AnswerBot

[59] C. D. Manning, H. Schütze, and P. Raghavan, *Introduction to information retrieval*. Cambridge university press, 2008.

[60] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[61] R. Mihalcea and P. Tarau, "Textrank: Bringing order into text," in *Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing , EMNLP 2004, A meeting of SIGDAT, a Special Interest Group of the ACL, held in conjunction with ACL 2004, 25-26 July 2004, Barcelona, Spain*. ACL, 2004, pp. 404–411.

[62] R. Likert, "A technique for the measurement of attitudes." *Archives of psychology*, 1932.

[63] Y. Zhang, D. Lo, X. Xia, and J. Sun, "Multi-factor duplicate question detection in stack overflow," *J. Comput. Sci. Technol.*, vol. 30, no. 5, pp. 981–997, 2015.

[64] M. Ahasanuzzaman, M. Asaduzzaman, C. K. Roy, and K. A. Schneider, "Mining duplicate questions in stack overflow," in *Proceedings of the 13th International Conference on Mining Software Repositories, MSR 2016, Austin, TX, USA, May 14-22, 2016*. ACM, 2016, pp. 402–412.

[65] L. Ponzanelli, A. Mocci, A. Bacchelli, M. Lanza, and D. Fullerton, "Improving low quality stack overflow post detection," in *30th IEEE International Conference on Software Maintenance and Evolution, Victoria, BC, Canada, September 29 - October 3, 2014*. IEEE Computer Society, 2014, pp. 541–544.

[66] S. Gottipati, D. Lo, and J. Jiang, "Finding relevant answers in software forums," in *26th IEEE/ACM International Conference on Automated Software Engineering (ASE 2011), Lawrence, KS, USA, November 6-10, 2011*, P. Alexander, C. S. Pasareanu, and J. G. Hosking, Eds. IEEE Computer Society, 2011, pp. 323–332.

[67] K. Luong, M. Hadi, F. Thung, F. Fard, and D. Lo, "Facos: Finding api relevant contents on stack overflow with semantic and syntactic analysis," *arXiv preprint arXiv:2111.07238*, 2021.

[68] B. Xu, Z. Xing, X. Xia, D. Lo, Q. Wang, and S. Li, "Domain-specific cross-language relevant question retrieval," in *Proceedings of the 13th International Conference on Mining Software Repositories, MSR 2016, Austin, TX, USA, May 14-22, 2016*. ACM, 2016, pp. 413–424.

[69] B. Sisman and A. C. Kak, "Assisting code search with automatic query reformulation for bug localization," in *10th Working Conference on Mining Software Repositories, MSR 2013, May 18-19, 2013, San Francisco, CA, USA*. IEEE Computer Society, 2013, pp. 309–318.

[70] M. Lu, X. Sun, S. Wang, D. Lo, and Y. Duan, "Query expansion via wordnet for effective code search," in *22nd IEEE International Conference on Software Analysis, Evolution, and Reengineering, SANER 2015, March 2-6,2015, Montreal, QC, Canada*. IEEE Computer Society, 2015, pp. 545–549.

[71] S. Haiduc, G. Bavota, A. Marcus, R. Oliveto, A. D. Lucia, and T. Menzies, "Automatic query reformulations for text retrieval in software engineering," in *35th International Conference on Software Engineering, ICSE 2013, May 18-26, 2013, San Francisco, CA, USA*. IEEE Computer Society, 2013, pp. 842–851.

[72] N. Zhang, Q. Huang, X. Xia, Y. Zou, D. Lo, and Z. Xing, "Chatbot4qr: Interactive query refinement for technical question retrieval," *IEEE Trans. Software Eng.*, 2020.

[73] K. Cao, C. Chen, S. Baltes, C. Treude, and X. Chen, "Automated query reformulation for efficient search based on query logs from stack overflow," in *43rd IEEE/ACM International Conference on Software Engineering, ICSE 2021, 22-30 May 2021, Madrid, Spain*. IEEE, 2021, pp. 1273–1285.

[74] A. Marcus, A. Sergeyev, V. Rajlich, and J. I. Maletic, "An information retrieval approach to concept location in source code," in *11th Working Conference on Reverse Engineering, WCRE 2004, November 8-12, 2004, Delft, The Netherlands*. IEEE Computer Society, 2004, pp. 214–223.

[75] O. Chaparro, J. M. Florez, U. Singh, and A. Marcus, "Reformulating queries for duplicate bug report detection," in *26th IEEE International Conference on Software Analysis, Evolution and Reengineering, SANER 2019, February 24-27, 2019, Hangzhou, China*. IEEE, 2019, pp. 218–229.

[76] M. Liu, X. Peng, A. Marcus, Z. Xing, W. Xie, S. Xing, and Y. Liu, "Generating query-specific class API summaries," in *27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/SIGSOFT FSE 2019, August 26-30, 2019, Tallinn, Estonia*. ACM, 2019, pp. 120–130.

[77] C. Wang, X. Peng, Z. Xing, Y. Zhang, M. Liu, R. Luo, and X. Meng, "Xcos: Explainable code search based on query scoping and knowledge graph," *ACM Transactions on Software Engineering and Methodology*, 2023.

[78] C. Wang, X. Peng, Z. Xing, and X. Meng, "Beyond literal meaning: Uncover and explain implicit knowledge in code through wikipedia-based concept linking," *IEEE Trans. Software Eng.*, vol. 49, no. 5, pp. 3226–3240, 2023.

[79] S. Xing, M. Liu, and X. Peng, "Automatic code semantic tag generation approach based on software knowledge graph," *Journal of Software*, vol. 33, no. 11, pp. 4027–4045, 2021.

[80] X. Peng, Y. Zhao, M. Liu, F. Zhang, Y. Liu, X. Wang, and Z. Xing, "Automatic generation of API documentations for open-source projects," in *IEEE Third International Workshop on Dynamic Software Documentation, DySDoc@ICSME 2018, Madrid, Spain, September 25, 2018*. IEEE, 2018, pp. 7–8.

[81] M. Liu, X. Peng, X. Meng, H. Xu, S. Xing, X. Wang, Y. Liu, and G. Lv, "Source code based on-demand class documentation generation," in *IEEE International Conference on Software Maintenance and Evolution, ICSME 2020, Adelaide, Australia, September 28 - October 2, 2020*. IEEE, 2020, pp. 864–865.

[82] M. Liu, X. Peng, A. Marcus, C. Treude, J. Xie, H. Xu, and Y. Yang, "How to formulate specific how-to questions in software development?" in *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/FSE 2022, Singapore, Singapore, November 14-18, 2022*. ACM, 2022, pp. 306–318.

[83] M. Liu, C. Zhao, X. Peng, S. Yu, H. Wang, and C. Sha, "Task-oriented ml/dl library recommendation based on a knowledge graph," *IEEE Transactions on Software Engineering*, 2023.

[84] L. Wang, X. Sun, J. Wang, Y. Duan, and B. Li, "Construct bug knowledge graph for bug resolution: Poster," in *39th International Conference on Software Engineering, ICSE 2017 - Companion Volume, May 20-28, 2017 , Buenos Aires, Argentina*. IEEE Computer Society, 2017, pp. 189–191.

[85] Y. Su, Z. Xing, X. Peng, X. Xia, C. Wang, X. Xu, and L. Zhu, "Reducing bug triaging confusion by learning from mistakes with a bug tossing knowledge graph," in *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2021, pp. 191–202.