



RAP4DQ: Learning to recommend relevant API documentation for developer questions

Yi Li¹ · Shaohua Wang¹ · Wenbo Wang¹ · Tien N. Nguyen² · Yan Wang³ · Xinyue Ye⁴

Accepted: 18 October 2021 / Published online: 29 November 2021

© The Author(s), under exclusive licence to Springer Science+Business Media, LLC, part of Springer Nature 2021

Abstract

Developers often face difficulties in using different API methods during the software development process. Answering API related questions on API Q&A forums often costs API development teams a lot of time. To help save time for API development teams, we propose a deep learning-based approach, namely RAP4DQ, to identify relevant web API documentation for developer's API related questions on API Q&A forums. RAP4DQ learns representation vectors for questions and API documentation separately using Gated Recurrent Unit (GRU) and adds different weights to reflect the various importance of varied API documents during training. RAP4DQ is designed to train on positive and negative samples with a loss function that minimizes the distances between questions and their relevant documentation, but maximizes the distances between questions and their irrelevant documentation. In the end, we construct a learning-to-rank layer to rank the API documentation based on learned representation vectors from GRUs. We have conducted several experiments to evaluate RAP4DQ on three popular and large API Q&A forums, Twitter, eBay, and AdWords. The results show that RAP4DQ can outperform all baselines by having a relative improvement up to 84.3% in terms of AUC. RAP4DQ can obtain a high AUC of 0.84, 0.88, and 0.94 on identifying relevant API documentation on Twitter, eBay, and AdWords, respectively.

Keywords Developer forums · Question answering · API documentation · Deep learning · Learning-to-Rank

1 Introduction

Accurately and effectively using Application Programming Interfaces (APIs) becomes critical in modern software development (Uddin and Khomh 2017). Recently, developer

Communicated by: Shaowei Wang, Tse-Hsun (Peter) Chen, Sebastian Baltes, Ivano Malavolta, Christoph Treude and Alexander Serebrenik

This article belongs to the Topical Collection: *Collective Knowledge in Software Engineering*

✉ Shaohua Wang
davidsw@njit.edu

Extended author information available on the last page of the article.

question and answering (Q&A) websites have become critical and essential online forums that allow developers to ask and answer questions regarding APIs, share and learn knowledge of using APIs, and have discussions on APIs (Mamykina et al. 2011). Quite often, API-related developer questions are answered by suggesting relevant API documentation that contains references for helping solve developer questions. Thus, automatically recommending relevant API documentation can help answer API-related developer questions (Li et al. 2018a).

Recently, two main types of developer Q&A websites have become popular among developers: general-purpose Q&A websites (e.g., Stack Overflow StackExchangeNetwork, 2020) and API Q&A forums that are run by API providers, e.g., Twitter API development forum (Twitter 2020). We use Stack Overflow (SO) as an example of general-purpose Q&A websites in the following discussion. The main differences between SO and API Q&A forums can be summarized as follows: (1) The questions on an API forum can be only relevant to the specific API, while SO accepts the questions relevant to any APIs; (2) Typically, an API forum is run by the API provider and has employees from the API development team to answer developer questions relevant to the API. The empirical investigation in existing empirical study (Li et al. 2020) suggests that about 87% of the questions that have answers on API forums are answered by API development teams. Developers tend to ask API-specific questions on API forums (Squire 2015), and API development teams on API forums can offer fast and right-to-the-point responses to API specialized questions (Venkatesh et al. 2016); and (3) SO provides incentives, e.g., badges and a voting system, to improve the published questions and their answers by allowing other developer users to edit them (Wang et al. 2018). A question or an answer can be modified multiple times on SO, while API Q&A forums often do not allow developers to modify others' questions or answers based on our observations. There have been some existing studies on Stack Overflow, such as (Silva et al. 2019; Huang et al. 2018; Gu et al. 2016; Li et al. 2018b). Despite the importance of API forums, little research has been focused on API forums. Therefore, our main goal in this paper is to automatically learn to recommend relevant API documentation to developer questions on API forums. In addition, we also evaluate our work on Java API related questions on Stack Overflow.

This goal is quite different from generating answers for open-domain questions. The "open-domain" here refers to the lack of the relevant context for any arbitrarily asked factual question. There are two main differences between them. The first difference is that open-domain question answering often requires the documentation used to answer questions should have relationships with each other. These relationships could be used for the existing models to generate the correct answer (often be a sentence) across documentation. But for our problem, the developers often do not know the order of the related API methods, and it does not influence the developers' understanding of the related API documentation. The second difference is that open-domain question answering often deals with short questions with short answers. But in our scenario, the API documentation could often be very long (over 1K words). Running the open-domain question answering approaches to our problem takes a huge amount of resources (over hundreds of GB memories) and a very long time (over several days) in our machine. For example, we tried to run several existing open-domain question answering approaches (He et al. 2020; Cao et al. 2020), the approach from He et al. (2020) we ran for one day without getting any results, and when we ran the other approach from (Cao et al. 2020), our machine reported the "out of memory" issue (our machine has 128GB memory).

As for our goal, to identify relevant API documentation to a given question, a wide range of existing approaches can be used, and they can be classified into the following two streams:

- **Unsupervised Approaches**, such as Rajaraman and Ullman (2011), Robertson et al. (2009), Mikolov et al. (2013), Lilleberg et al. (2015), Brokos et al. (2016), and Kusner et al. (2015). Quite often, this type of approaches first generate vectors for documentation and questions, then calculate the similarities among them. However, they do not work well if the wordings in documentation and questions are very different. Typically, API documentation and developer questions are in different wordings (Li et al. 2018a), which makes it challenging for unsupervised approaches to search relevant API documentation for a question.
- **Supervised Approaches**, such as Sutskever et al. (2014), Luong (2015), and Li et al. (2018a). To improve the matching of different vocabularies in documentation and questions, some supervised approaches, such as Xue et al. (2008) and Nicosia et al. (2015), employ statistical and language models to identify relations among documentation and questions by relying on hand-crafted learning features. Some other approaches, such as Li et al. (2018a), Sutskever et al. (2014), and Palangi et al. (2016), adopt deep learning to automatically learn relations among documentation and questions. However, the existing approaches are often trained with only positive samples that are pairs of the question and its relevant document. In an API documentation list, some API documentation can have very similar descriptions, which makes only training with positive samples insufficient.

To overcome the aforementioned limitations, we propose a deep learning-based approach to identify relevant API documentation from the API documentation list to answer developer questions. Specifically, RAP4DQ first learns word embeddings for API documentation and questions using word2vec (Mikolov et al. 2013). Second, to add more discriminative power into RAP4DQ, we build RAP4DQ to train on positive and negative samples. For a given question, q , a positive sample is a document that is relevant to q , and a negative sample is a document that is irrelevant to q . Third, due to the high heterogeneity in wordings of an API document and a developer question, we learn the representation vectors for documentation and questions separately from their word embedding vectors using Gated Recurrent Unit (GRU) (Cho et al. 2014). Furthermore, we add trainable weights to the API documentation, as distinct API documentation can have different importances on an API forum. Last, RAP4DQ classifies if an API document is relevant to a question or not using SoftMax (Bishop 2006). To train RAP4DQ, we develop a new loss function that aims to reduce the distance between questions and their relevant API documentation but increase the distance between questions and their irrelevant API documentation. To get a higher quality of recommendations, after our relationship learning approach, we also added a learning-to-rank layer to rank the API documentation list for each API related question. By using both the representation vectors generated by GRUs for questions and documentations together as the relationship feature vectors, the learning-to-rank model can help to rank the API documentation to generate the higher quality recommendations for API related questions.

We have conducted several experiments to evaluate RAP4DQ on three popular API Q&A forums, Twitter (Twitter 2020), eBay (eBay 2020), and Google AdWords (Adwords 2020), and their API documentation. We compare RAP4DQ with 17 state-of-the-art approaches, including some unsupervised approaches, such as TF-IDF (Rajaraman and Ullman 2011), BM25 (Robertson et al. 2009), Word2vec (Mikolov et al. 2013), IDFword2vec

(Lilleberg et al. 2015), Word Mover Distance (WMD) (Kusner et al. 2015), and CenIDF + K-Nearest Neighbors (KNN) + WMD (Brokos et al. 2016) (This baseline contains 4 different approaches.), and some other learning-based approaches, such as seq2seq (Sutskever et al. 2014), seq2seq with attention (Luong 2015), QDLinker (Li et al. 2018a), ELMo (Peters et al. 2018), Bert (Devlin et al. 2018), GPT-2 (Radford et al. 2019), BIKER (Huang et al. 2018), and CROKAGE (Silva et al. 2019). Our empirical results show that RAP4DQ can outperform all 17 baselines on recommending relevant API documentation to API related questions on API forums. RAP4DQ can relatively improve the baselines by up to 84.3% in terms of AUC. RAP4DQ can achieve a high AUC of 0.84, 0.88, and 0.94 on Twitter, eBay, and AdWords forms, respectively.

We also evaluate RAP4DQ on Stack Overflow. Specifically, We compare RAP4DQ with four Java APIs question-answering approaches (Silva et al. 2019; Huang et al. 2018; Gu et al. 2016; Li et al. 2018b). The results show that RAP4DQ outperforms all baselines by improving up to 117.1% in terms of AUC on Stack Overflow.

In this paper, we make the following contributions:

- **Learning to identify relevant web API methods for developer questions on API Q&A forums.** Little research has been conducted on API specialized Q&A forums, even though they are one of the popular types of Q&A websites. To the best of our knowledge, our work is the first to automatically recommend relevant API documentation to answer developer questions on API Q&A forums that have different characteristics from the ones of Stack Overflow.
- **A new deep learning-based approach that identifies relevant API methods in the documentation.** We build a new Gated Recurrent Unit (GRU) based approach that can learn different representation vectors for documentation and questions, train on positive and negative samples, differentiate the importance of various API documentation, employ a new loss function that trains RAP4DQ towards reducing distances between questions and their relevant documentation, but increasing distances between questions and their irrelevant documentation, and a learning-to-rank technique to improve the recommendation accuracy.
- **An extensive comparative evaluation and in-depth analysis.** Through a series of empirical evaluations, our results show that RAP4DQ outperforms the state-of-the-art baselines on both API forums and Stack Overflow datasets. We also conducted a sensitivity analysis of the impact of different factors on RAP4DQ. Furthermore, we built a large labeled dataset as a benchmark for recommending relevant documentation on API forums. Our replication package is available publicly (Rap4DQ Replication 2020).

2 Motivation and Approach Overview

2.1 Motivating Example

From the results of the existing study (Li et al. 2020), we can see that about 63% of the questions were answered by using API methods in the API documentation. This high percentage shows that a lot of developer questions in the API Q&A forums are related to API methods. Here is a real developer question as follow:

Figure 1 shows a real developer question example from the eBay forum. The question is asking about how to use the product ID to get the description of the item that he is finding.

andrzejborucki asked · May 31 '16 at 8:06 AM

How to find product descriptions by ID?

On [eBay_APICall_CodeSamples][1] is example for searching by keywords. Line `port.findProducts(request)` prints xml to console. How do not print? How make example for finding not ID by keywords, but finding product descriptions by ID? For example: print descriptions of 381577929800 - 381577929809 ? [1]: https://github.com/eBayDeveloper/eBay_APICall_CodeSamples/blob/master/Shopping/FindProducts/FindProducts-SOAP-JAXWS-JAVA/src/com/ebay/test/shopping/FindPopularSearches.java

finding api

Comment

2 Answers · Write an Answer

corc-ches answered · Jun 07 '16 at 11:09 PM

If you have the ItemID, I believe you can use Trading API's GetItem call. Document at this link: <http://developer.ebay.com/Devzone/XML/docs/Reference/eBay/GetItem.html>

Comment · Share

Fig. 1 An Example of a Real Developer Question (Ebay 2019) with its Answer from the eBay Development Team. The Relevant API Method is Underlined in Red

And the development team replied to him that he could use the Trading API's `GetItem` call to achieve the function that he wants. From the answer that the development team provided, we can easily figure out that the API method `GetItem` is closely relevant to this developer question. However, some API methods are similar to `GetItem` that may confuse the developers. For example, the API methods `GetSingleItem` and `GetMultipleItems` are very similar to `GetItem`. All of these three API methods have similar names and are used to get the product information, but the API method `GetItem` can get private information while the other two can only get public information for the product. So the `GetItem` is often been used for product owner and the `GetSingleItem` and `GetMultipleItems` are often been used for customers. With this example, we come up with the following observations:

Observation 1 The vocabulary in the API documentation and API related questions are different by using different words, using different sentence structures, and using different sentence tones. For example, in Fig. 1, in the question, the developer asks questions by using the sentence like "How do not print?" while in the documentation, it uses descriptions to explain the API method, like "Use this call to retrieve the data for a single item listed on an eBay site." Existing Q&A approaches cannot deal with it well because they often regard the question and documentation in the same domain with the same vocabularies. For example, the QDLinker (Li et al. 2018a) cannot find the related API documentation for this question because it considers all questions and documentation with the same vocabulary to find the similarity between questions and documentation.

Observation 2 There are some API methods with the similar name but the functionality of them are different such as the `GetItem` and `GetSingleItem` in Fig. 1. These similar API methods sometimes even have similar documentation content with little difference. The existing Q&A approach cannot figure out the differences between them. For example, the QDLinker (Li et al. 2018a) encodes the documentation directly without strategies to separate these two similar API methods. It causes the QDLinker cannot correctly recommend related API documentation.

2.2 Key Ideas

According to the above observations, we have the following key ideas:

- Idea 1. Learning Similarity Between Developer Questions and API Documentation**
To catch the relationship between the developer questions and API documentation, we would like to use a deep learning framework to encode both the questions and the documentation separately. In the encoded vector space, The questions and the relevant documentation representation vectors should have a smaller distance while the question and the irrelevant documentation representation vectors should have a bigger distance.
- Idea 2. Considering Negative Samples.** In order to reduce the influence of the similar API calls, we would like to bring in the idea of using irrelevant API documentation as negative samples to enhance the deep learning framework to let the model can find out the most relevant API documentation from a group of similar API documentation.
- Idea 3. Distinguishing Documentation Importance** Even we use negative samples to distinguish similar API calls, sometimes the similar API documentation is still hard to be separated when making the recommendation. To improve the accuracy of the recommendation and reduce the influence from similar API calls, we would like to add weights for different API documentation to specialize the importance of each API documentation.
- Idea 4. API documentation Ranking** To solve the problem that a question may have more than one related API documentation, we would like to use a ranking technique to rank the API documentation. The ranking technique can take the generated representation vectors from GRUs with some processings as the feature vectors to learn and predict the ranked API documentation list as the final recommendation of RAP4DQ.

2.3 Overview of RAP4DQ

To overcome the drawbacks of existing approaches, we propose a deep learning-based approach to identify relevant APIs for a given question. RAP4DQ has the following main steps as illustrated in Fig. 2:

- Step 1. Learning Word Embeddings.** We first pre-process API documentation, developer questions, and their answers by removing stop words and special characters in textual description and extracting method names from code. Then, we learn word embeddings of all words in the pre-processed API documentation, developer questions, and their answers using word2vec (Mikolov et al. 2013). Each question and API documentation is represented as a sequence of word vectors.

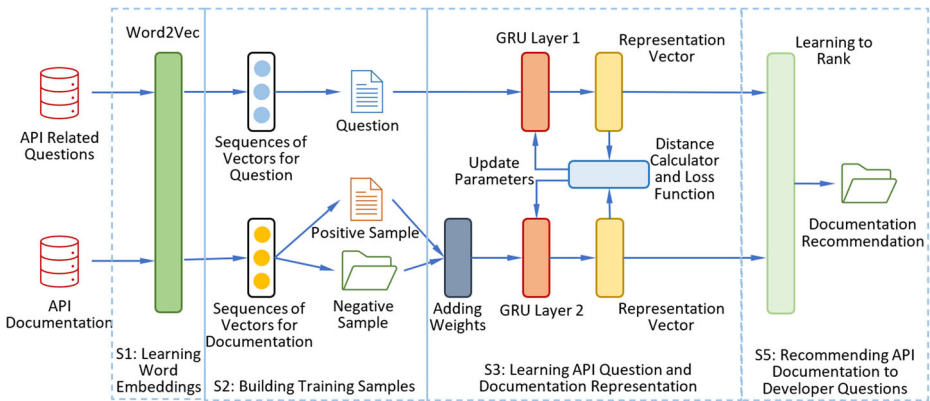


Fig. 2 The Overview of RAP4DQ

- Step 2. Building Training Samples.** A developer question can be answered by one or more API methods (i.e., each method is described on a web page). We first create a one-to-one mapping $q \rightarrow m$ between a question, q , and a relevant API method, m . The same question can have multiple one-to-one mappings. Second, for each $q \rightarrow m$, we randomly select a set of irrelevant API documents (denoted as NS) for the q as negative samples, so we form a tuple $\langle q \rightarrow m \rightarrow NS \rangle$ for the q .
- Step 3. Learning API Question and Documentation Representations.** We build a two-parallel-layer architecture to generate question and API document representations. We use one Gated Recurrent Unit (GRU) (Cho et al. 2014) layer to generate one representation vector for a developer question, q , and another GRU layer to generate representation vectors for API documentation relevant or irrelevant to q . As various API documentation have different importances on an API forum, we add weights to distinct documentation and generate documentation representation vectors using GRU. To get the most suitable parameters in GRUs during training, we firstly use question and API documentation representation vectors to calculate the cosine similarity as the distance between a question and the API documentation. Then we come up with a new loss function that makes the model reduce the distance between a question and its relevant documentation but increases the distance between the question and its irrelevant documentation. Based on the loss function, the parameters in GRUs will be updated after each running of the model during training.
- Step 4. Recommending API Documentation to Developer Questions.** We create a learning-to-rank layer on top of the deep learning-based model generated features to rank the API documentation for each question to improve the accuracy of the API documentation recommendation. Given a list of API documentation D for a developer questions q , we apply a scoring function to generate the ranking score $S(q_i, d_j)$ for each documentation d_j in D by using the representation vectors q_i and d_j generated from the well-trained deep learning-based model in step 3 and rank the relevant documentation also based on this score. We finally recommend the ranked documentation list for the developer question q . The detailed ways of calculating the $S(q_i, d_j)$ can be found in the approach section.

3 Approach

In this section, we delve into the details of each main step of RAP4DQ.

3.1 Step 1: Learning Word Embeddings

In this step, we would like to use the existing language model to generate the word embedding for each word in the questions, answers, and API documentation. The input of this step is the textual part in developer questions, their answers, and API documentation, including description and code. And the output of this step is the word embeddings for all words in questions, answers, and API documentation.

We take the following steps to process the textual description and code in the API documentation, developer questions, and their answers: First, we remove any special characters (e.g., \$ and #) that are not commonly used in natural languages as suggested in NLTK (2020) and stop words from the textual description. Second, we extract keywords from the code, such as a method name, and remove all other parts of the code.

After data cleaning, we learn the word embeddings for each word in the API documentation (D), API related questions (Q), and their answers (A) using word2vec (Mikolov et al. 2013), a neural network that takes a text corpus as an input and generates a set of feature vectors for words in the corpus. This step takes all words from D , Q , and A of each API forum.

For example, in Fig. 3, we process the sentence “*Unfortunately I only find the specific criterias for the dsas (<https://developers.google.com/adwords/api/docs/guides/dynamic-search-ads?hl=en>) and not the option 'All websites'.*” into the following sentence “*Unfortunately I only find the specific criterias for the dsas [https developers . google . com adwords api docs guides dynamic-search-ads hl en](https://developers.google.com/adwords/api/docs/guides/dynamic-search-ads?hl=en) and not the option ' All websites ' .*”, then, we use space to separate each element in this cleaned sentence and learn word embeddings.

3.2 Step 2: Building Training Samples

After having the word embeddings, we would like to build the positive samples and negative samples for each question to prepare to do the training in the next step. The input of this step is the word embeddings, the developer questions, and the API documentation. We regard the generated pairs of the question, the positive sample, and the negative samples as the training samples, and it is the output of this step.

Specifically, first of all, we use the generated word embedding in the last step to replace each word with its word vector in all questions and API documentation to make them can be used by the deep learning model in the next step. The questions and API documentation will become a sequence of vectors. Next, before we created the training samples, we found that a developer question can be answered by one or more API method documents. For example, Fig. 3 shows that there are two API documents, *Webpage* and *WebpageParameter*, that are relevant to the question in Fig. 3. We first create a one-to-one mapping between a question, q , and a relevant API method document as a positive sample, m , denoted as $q \rightarrow m$. The same question can have multiple one-to-one mappings. For each mapping $q \rightarrow m$, we randomly select N irrelevant API documents that are not used in the answers to q as negative samples (denoted as NS), so we form a training tuple $\langle q \rightarrow m \rightarrow NS \rangle$ for q . N is a parameter that can be learned empirically.

How to create dynamic Targeting 'All websites' 20 views Subscribe ✕



bitsmugler

to adwor...@googlegroups.com

5/8/17

Hi there

I try to create the dynamic targeting 'All websites' via AdWords API. Unfortunately I only find the specific criterias for the dsas (<https://developers.google.com/adwords/api/docs/guides/dynamic-search-ads?hl=en>) and not the option 'All websites'. Could you help me?

Thanks.

Best, patrick



Shwetha Vastrad (AdWords API Team)

to adwor...@googlegroups.com

5/8/17

Hi Patrick,

You'll need to create a [Webpage](#) criterion with the following [WebpageParameter](#) to create an auto target which corresponds to "All websites":

```
<parameter>
  <CriterionParameter.Type>WebpageParameter</CriterionParameter.Type>
  <criteriaName>*</criteriaName>
</parameter>
```

A null list of [conditions](#) in the WebpageParameter means that all webpages of the campaign's website are targeted.

Regards,
Shwetha, AdWords API Team.

Fig. 3 An Example AdWords Question (AdWords 2019) and Its Answer from the AdWords API Development Team. The Relevant API Methods are Underlined in Red

For example, in Fig. 3, we create two one-to-one mappings for the question: *question* → *Webpage* and *question* → *WebpageParameter*. Then, we randomly select N other AdWords API documents as negative samples for each mapping. Then for each mapping, we replace all words with word vectors in the question, positive sample, and negative samples to create a training sample.

3.3 Step 3: Learning API Question and Documentation Representations

With the training samples generated in the last step, in this step, we would like to use training samples as input to train our deep learning model to learn the representation vectors for questions, positive samples, and negative samples. In order to make the generated representation vectors have higher accuracy, we propose a new loss function for our representation learning model to update the model parameters during the training. These representation vectors are the output of this step and will be used in the next step.

Within this step, because the language structures and vocabularies of API specialized questions and API documentation can be very different, we use two Gated Recurrent Unit (GRU) layers (Cho et al. 2014) to generate representation vectors for API questions and documentation. Specifically, we first learn representations for API questions. After step 2, a

question, q , is represented as a sequence of word vectors, and we use a GRU layer to learn the question representation vector by using the sequenced word vectors of q .

Second, we apply another GRU layer on word vectors of API documentation to generate representation vectors for the API documentation. Based on existing study (Li et al. 2020), we find that the API documents can have different numbers of times they are mentioned on an API forum (e.g., a large portion of documents are not even mentioned once.), which indicates that distinct API documents have different importances. To reflect the importance of different API documents, we assign a weight to each API document (i.e., including a m and NS). For the i -th API document, we calculate the weight, w_i , as follows:

$$w_i = \begin{cases} p_1 * \frac{T_i}{T_{max}} & T_i > 0 \\ 1 & T_i = 0 \end{cases} \tag{1}$$

Where w_i is the weight for the i -th API document, p_1 is an automatically trained parameter, T_i is the number of times the i -th API document mentioned in an API forum, T_{max} is the maximum number of times an API document among all API documents mentioned in the API forum.

To get the best parameters in the GRUs for our tasks, once we get representation vectors for questions and API documentation, we use cosine similarity to calculate the distances between questions and API documentation and create a loss function to update the GRUs parameters to increase the total performance.

Given a set of training tuples, for each tuple, $\langle q \rightarrow m \rightarrow NE \rangle$, that has a question (q), a relevant API document (m) to q , and a set of irrelevant API documents (NS), we aim to create a loss function that leads to reduce the distance between q and m , but increases the distance between q and every API document in NS . We formulate our aim into the following optimization problem:

$$L = \min -\log \prod_{i=1}^T P(m_i|q_i) \tag{2}$$

where T is the total number of training tuples, $1 \leq i \leq T$, $P(m_i|q_i)$ is the probability of a relevant API document given the i -th question. We further expand (2) into the following one containing a loss function:

$$\min \sum_{i=1}^T -\log \left(\frac{e^{S(V_i^q, V_i^m)}}{e^{S(V_i^q, V_i^m)} + \sum_{j=1}^{|NE|} e^{S(V_i^q, V_i^{NE_j})}} \right) \tag{3}$$

Where $S(V_i^q, V_i^m)$ is the cosine similarity between a representation vector of i -th question and its relevant API document representation vector, $|NE|$ is the number of negative samples, $S(V_i^q, V_i^{NE_j})$ is the cosine similarity between a representation vector of i -th question and a representation vector of j -th irrelevant API document in NS to the i -th question.

During training, for a training tuple containing the mappings, $\langle q \rightarrow m \rightarrow NE \rangle$, we pass a question (q) into one GRU layer and both positive and negative samples (m and NS) into another GRU layer and for the training target, we set the value 1 for relevant and 0 for irrelevant.

During the prediction, our model generates data tuples by using $\langle q \rightarrow D \rangle$ where q for an incoming question q and D for one of the API documentation. And then, our well-trained model could learn the representation vectors for both q and D that are the output for this step and will be used in the next step.

3.4 Step 4: Recommending API Documentation to Developer Questions

After we have the deep learning-based model from step 1 to step 3, during prediction, we use the generated representation vectors for questions and API documentation in the last step as the input for this step. We will generate a ranked documentation list as the recommendation for developers in this step as the output.

In particular, we concatenate V_i^q and V_j^d that are representation vector of i -th question q_i and the j -th API documentation into one feature vector V_f . V_f represents the relationship between the question q_i and the API documentation r_j . Then we feed the V_f to a learning-to-rank schema. The learning-to-rank schema can use different features to do the ranking and can automatically learn the ways of combining these features. In this step, we train the learn-to-rank model using LambdaMART (Wu et al. 2010) and learn a scoring function as follow:

$$S(q_i, r_j) = \sum_{k=1}^K w_k * f_k(q_i, r_j) \quad (4)$$

where each feature $f_k(q_i, r_j) \in V_f$ measures a relationship between the question and the API documentation. w_k is a trainable weight for the k -th feature. We train the ranking model using Ranklib (Ranklib 2020), a java library of learning-to-rank algorithms.

With the scoring function above, we could measure the relevance between the question q_i and the API documentation d_j by using the score $S(q_i, d_j)$ and use this score to rank the API documentation. We will provide a ranked list of API documentation as the final recommendation of our model.

4 Experimental Setup

We conduct several empirical experiments to evaluate RAP4DQ on a desktop with a 4-core Intel CPU and a single GTX Titan graphics card.

4.1 Research Questions

Through our empirical experiments, we aim to answer the following research questions:

RQ1. Unsupervised Question Answering Comparative Study on API Forums. How well does our approach perform in comparison with existing state-of-the-art unsupervised approaches?

RQ2. Supervised Question Answering Comparative Study on API Forums. How well does our approach perform in comparison with existing state-of-the-art supervised approaches?

RQ3. Supervised Question Answering Comparative Study on Stack Overflow (SO). How well does our approach perform in comparison with existing approaches on Java API related questions on SO?

RQ4. Sensitivity Analysis. How do various factors affect the overall performance of our approach?

4.2 Data Collection and Processing

We evaluate RAP4DQ on different datasets: three API forum datasets and one Java API Stack Overflow dataset (Li et al. 2018a).

API Forum Datasets: We use the three forum datasets (Twitter, eBay, and AdWords API forums) from Li et al. (2020). Each dataset contains the questions, the answers, and the relevant API methods of a form. Table 1 shows descriptive statistics of the three forum datasets.

Golden Set The collected API forum datasets do not have labels. Therefore, we build a golden set from them to evaluate approaches. In each forum dataset, a question is associated with their relevant API methods, and each API method has a document describing the API method. Specifically, during the labeling, each time, we randomly picked one question from the questions having at least one answer. Given a question, q , we manually study each answer from the API development team to q . If the answer to q cannot be answered using API documentation, we discard q and move to the next question. If the answer contains any API method names, we label the API methods as relevant to q .

We conduct manual labeling, as the API methods mentioned in an answer are (1) quite often implicit and require humans to extract them, and (2) not relevant anymore to the question due to API evolution (e.g., API deletions). In (2), the API development team usually directs the questioner to another API method in a newer version of the API. For simplicity, we use the most up-to-date version of the API documentation to answer the questions. If any API methods mentioned in an answer to a question do not exist anymore in the newer API version, we simply discard the question, as old web API documentation is not accessible anymore.

To avoid biases in the manual labeling, We recruited three participants to conduct the labeling. We use the majority-win rule to decide the final label for a question. The majority-win rule means that among three participants if at least two participants agree with one label, this label will be the final label of the question. Overall, we obtained a high inter-rater agreement of 0.85, indicating that the quality of our labeling is good and our golden set is good for evaluation. The reason of having a high inter-rater agreement is because in most cases, participants simply extracted the relevant API names from the answers provided by the official API development team. Therefore, the accuracy of the labeling is high in this situation. In total, due to the limited manpower, we collected 1,000 questions with labels for each forum. Because the dataset is built with real data and the biases have been controlled at an acceptable level, the evaluations on this dataset are reliable.

Stack Overflow Dataset: We evaluate RAP4DQ on a Java API Stack Overflow dataset (Li et al. 2018a). Table 2 shows descriptive statistics of the Stack Overflow dataset.

Table 1 Statistics of our collected API forum dataset

Statistics	Twitter	eBay	AdWords
# of API docs	410	457	2,774
# of Questions	16,874	6,204	23,731
# of Total Words	8.05M	5.51M	8.39M
# of Unique Words	13k	12k	18k
Average # of Words in an API doc	763	755	516
Average # of Words in a Question	459	368	326
Max Number of Words in an API Doc	3,991	3,986	6,578
Max Number of Words in a Question	1,789	2,431	1,342

Table 2 Statistics of our collected stack overflow dataset

Statistics	Stack overflow
# of API docs	2608
# of Questions	16876
# of Total Words	6.72M
# of Unique Words	15k
Average # of Words in an API doc	196
Average # of Words in a Question	10
Max Number of Words in an API Doc	3992
Max Number of Words in a Question	40

Dataset Size: During the training of RAP4DQ, there are 11K question-document pairs generated from each API forum dataset. Within these pairs, they contain 9K+, 8K+, and 11K+ unique words on Twitter, eBay, and AdWords datasets that are at the same level as the Stack Overflow dataset (15K+). We applied the Dropout layer to reduce the overfitting. We also evaluate RAP4DQ on the Stack Overflow dataset.

4.3 Evaluation Metrics

We use Area Under the ROC Curve (AUC) to evaluate the effectiveness of an approach on identifying relevant APIs to developer questions. The AUC is calculated as follows:

$$AUC = \frac{1}{|Q|} \sum_{q \in Q} \frac{1}{|I_q^+||I_q^-|} \sum_{i \in I_q^+, j \in I_q^-} \delta(i > j) \tag{5}$$

where Q is the total number of questions, I_q^+ is *True Positive (TP) + True Negative (TN)*, I_q^- is *False Positive (FP) + False Negative (FN)*, and $\delta(i > j)$ is an indicator function that takes the value 1 if and only if $(i > j)$ is true. In other words, we are counting the fraction of times that the 'observed' documentation i is preferred over 'non-observed' documentation j . As our goal is to recommend a set of relevant APIs that can be used to answer a question, the ranking of the recommended APIs is not important, but the relevance of an API to a question is important. Therefore, we use AUC as our training target to maximize the possibility of a relevant API in our recommended set. An AUC value of 0.5 implies that a classifier is no better than random guessing. A larger AUC value indicates a better performance.

In our evaluation, even though we use AUC as the main target, we also use some other evaluation metrics to help do the evaluation together. These metrics include Accuracy, MAP, MRR, Precision at top 20, Recall at top 20, NDCG. We use the formulas in Table 3 to calculate these metrics. Among the formulas, TP is true positive; TN is true negative; FP is false positive; FN is false negative; n is the total number of results; k is the current rank in the list; $rel(k)$ is an indicator function equaling to 1 if the item at rank k is true, and to zero otherwise; Q is the total number of classification types; r_i is the score of the result at position i ; R_k is the rank of the actual true label in the resulting list up to the position k ; $rank_i$ refers the rank position of the first relevant document for the i -th question; $AvgP$ is calculated as $AvgP = \sum_{k=1}^n P(k)rel(k)$; DCG_k is calculated as $DCG_k = \sum_{i=1}^k \frac{r_i}{\log_2(i+1)}$; and $IDCG_k$ is calculated as $IDCG_k = \sum_{i=1}^{|R_k|} \frac{2^{r_i-1}}{\log_2(i+1)}$.

Table 3 The evaluation metrics

Measure	Definition	Description
Accuracy	$Accuracy = \frac{TP+TN}{TP+FP+FN+TN}$	The degree of closeness to the true label
MAP	$MAP = \frac{\sum_{q=1}^Q AvgP(q)}{Q}$	Mean of the average precision scores for each query
MRR	$MRR = \frac{1}{ Q } \sum_{i=1}^{ Q } \frac{1}{rank_i}$	Average of the reciprocal ranks of results
NDCG	$nDCG_k = \frac{DCG_k}{IDCG_k}$	All queries can be averaged to obtain a measure of the average performance of a ranking algorithm

4.4 RQ1. (Unsupervised Question Answering Comparative Study on API Forum.) Analysis Approach.

4.4.1 Comparison baselines

We build the following unsupervised 12 baselines to identify a set of relevant API methods to a question:

- *TF-IDF* (Rajaraman and Ullman 2011): It uses TF-IDF to generate vectors for both questions and documentation, and calculates the cosine similarity between the questions and the API documentation.
- *Okapi BM25* (Robertson et al. 2009): It uses Okapi BM25 to get vectors for both questions and documentation, and calculates the cosine similarity between the questions and the API documentation.
- *word2vec* (Mikolov et al. 2013): It uses word2vec to get word vectors and utilizes them to build question and documentation vectors to calculate the cosine similarity.
- *IDF + Word2vec* (Lilleberg et al. 2015): The baseline generates word vectors using word2vec, and assigns IDF scores to word vectors. Second, it uses the weighted word vectors of the questions and the API documentation to calculate the similarity.
- *Word Mover Distance (WMD)* (Kusner et al. 2015): WMD is a very powerful way to calculate the similarity between the text documentation. We directly apply WMD to calculate the similarity between the questions and the API documentation.
- *Centriod IDF + K-nearest neighbors (KNN)* (Brokos et al. 2016): It first has the centroid IDF vectors, and then uses KNN to find the relevant API documentation to questions.
- *Centriod IDF + Approximate Nearest Neighbors (ANN)* (Brokos et al. 2016): This baseline is similar to the above one, except that it uses ANN instead of KNN.
- *Centriod IDF + KNN + WMD* (Brokos et al. 2016): It uses the above method, *Centriod IDF + KNN*, to get top 50 documents for a question. Then, it applies WMD to re-rank the top 50 documents.
- *Centriod IDF + ANN + WMD* (Brokos et al. 2016): This baseline is similar to the above one, except using ANN instead of KNN.
- *Pre-trained ELMo* (Peters et al. 2018): This baseline uses the pre-trained ELMo model from Peters et al. (2018) to learn word vectors and directly calculates the cosine similarity between the questions and API documents.
- *Pre-trained Bert* (Devlin et al. 2018): This approach is designed to use the pre-trained Bert model and directly calculates the cosine similarity between the questions and API documents.

- *Pre-trained GPT-2* (Radford et al. 2019): This approach is designed to use the pre-trained GPT-2 model and directly calculates the cosine similarity between the questions and API documents.

4.4.2 Model Training and Parameters Tuning

In this RQ, we evaluate the baselines on three API forum datasets. The input of the model in this RQ is the question title with the question body, and the expected output is the related API documentation.

Within this RQ, RAP4DQ requires learning word vectors for questions and API documents (e.g., using TF-IDF or word2vec). We use all words from the questions in the training dataset and their answers on each forum and all API documentation to learn word vectors. When making a prediction, if there come the unseen words, we use zero-vector to represent it. With the learned word vectors, we run all applied baselines and RAP4DQ on the well-labeled golden set. We tune all baselines to achieve the best results on our dataset. Due to the page limit, we skip the explanation for baselines that do not need tuning.

Several baselines use KNN and ANN as a component. We use the package *sklearn* (Scikit-learn 2020) and the package *annoy* (Annoy 2020) to automatically tune KNN and ANN for the best results on our dataset, respectively. Several baselines use word2vec to obtain word vectors. For word2vec, we do not directly use the pre-trained word2vec because the word distribution in API related questions and documentation would be different from other topics, and that makes the pre-trained word2vec cannot encoding the information in our problem well. As for training our own word2vec, we set the parameters as *size* = 300, *window* = 10, *sample* = $1e - 4$, *threads* = 10 and all other parameters as default values introduced in *keras* (Keras 2019). We use the same Word2vec setting for all relevant baselines and RAP4DQ. The word2vec is trained on all words and sentences in questions and documentation except the 1,000 API related questions that we picked and labeled to do the evaluation. As for the pre-trained ELMo, Bert, and GPT-2, we directly use their pre-trained model to select relevant API documentation. The usage of a pre-trained model can be regarded as the unsupervised approach.

For RAP4DQ, we tune the following hyperparameters: learning rate [0.01, 0.05, 0.1, 0.15], epoch size [150, 200, 250, 300], and batch size [16, 32, 48, 64].

4.5 RQ2. (Supervised Question Answering Comparative Study on API Forum) Analysis Approach.

4.5.1 Comparison baselines

We build the following six supervised baselines to identify a set of relevant APIs to a question:

- *Seq2seq* (Sutskever et al. 2014): This baseline uses Seq2seq to learn the relations between questions vectors to API documentation vectors and translates a given question to a vector, v . Then, it calculates the similarity between v and every API documentation to identify the relevant ones.
- *Seq2seq attention* (Luong 2015): This baseline adds an attention layer to the above Seq2seq.

- *QDLinker* (Li et al. 2018a): It mainly uses a Deep Neural Network (DNN) to detect relevant documentation, uses a learning-to-rank model to re-rank the detected documentation, and recommends the most relevant documentation.
- *Re-trained ELMo* (Peters et al. 2018): Unlike the unsupervised baseline *Pre-trained ELMo*, this baseline adopts the architecture of ELMo in Peters et al. (2018) and re-train the ELMo using our forum datasets with true labels to learn embeddings and relevance between questions and API documents.
- *Fine-tuned Bert* (Devlin et al. 2018): This baseline adopts the pre-trained Bert model and uses our forum datasets to fine-tune the pre-trained Bert model. The fine-tuning process is supervised and updates the word embeddings in the pre-trained Bert model to the downstream tasks.
- *Fine-tuned GPT-2* (Radford et al. 2019): This approach is similar to the above one, except that this one uses the pre-trained GPT-2 and fine-tunes it.
- *BIKER* (Huang et al. 2018): This approach is an API recommendation approach to automatically recommend relevant APIs for a programming task described in natural language.
- *CROKAGE* (Silva et al. 2019): This approach takes the description of a programming task and provides a comprehensive solution for the task.

4.5.2 Model Training and Parameters Tuning

Similar to RQ1, the input in this RQ is the question title and the question body, and the expected output is the related API documentation. The only extra thing is that in this RQ, we use 80% of the golden set to train supervised baselines, 10% to tune the model, and another 10% for testing.

word2vec in this RQ we keep the same parameters as RQ1 including $size = 300$, $window = 10$, $sample = 1e - 4$, $threads = 10$ and all other parameters as default values introduced in *keras* (Keras 2019).

For seq2seq and seq2seq attention baselines, we tune hyperparameters via grid search on our dataset. We test the learning rate with the values of [0.001, 0.005, 0.01]. We test the following values [200, 250, 300] for the epoch size and [16, 32, 48] for the batch size. We set all other parameters as default values introduced in *keras* (Keras 2019).

For the QDLinker, we tune hyperparameters during training on our dataset to achieve the best results. We test the following filter numbers [32, 64, 96] and sizes of filter [1, 2, 3].

For ELMo, we tune hyperparameters for learning rate [0.003, 0.004, 0.005], λ [0.0005, 0.001, 0.003], dimension size [200, 250, 300], and batch size [16, 32, 64]; For Bert, we fine-tune the pre-trained model by setting the epoch size [3, 4, 5], batch size [16, 32, 64], and learning rate [$3e - 4$, $1e - 4$, $5e - 5$]; For GPT-2, we fine-tune the pre-trained model by setting the learning rate [$1e - 5$, $2e - 5$, $3e - 5$] and steps [500, 1000, 1500]. We do not do the re-train on Bert and GPT-2 because re-train them requires a huge amount of machine resources, and our server cannot handle the computation cost. But for ELMo, we can run it on our machine perfectly.

In the process of generating the answers for the question, the CROKAGE uses RACK (Rahman et al. 2016), BIKER (Huang et al. 2018) and NLP2API (Rahman and Roy 2018) to generate scores for API methods and then use the scores to help generate paragraphs for answering the questions. To make a fair comparison with RAP4DQ, we directly use these scores to rank relevant API documents. The output of BIKER contains the relevant API names. We use these API names as the recommended API methods to rank relevant API documents. To tune the parameters of the model, because in BIKER and CROKAGE there

is not a clear instruction, we simply tune the most common parameters such as learning rate [0.001, 0.005, 0.01], dimension size [200, 250, 300], and batch size [16, 32, 64].

For RAP4DQ, we follow the same process of tuning in RQ1.

4.6 RQ3. (Supervised Question Answering Comparative Study on Stack Overflow) Analysis Approach.

In this RQ, we bring in a new dataset for Java API from Stack Overflow.

4.6.1 Comparison baselines

- *BIKER* (Huang et al. 2018): This approach is an API recommendation approach to automatically recommend relevant APIs for a programming task described in natural language.
- *CROKAGE* (Silva et al. 2019): This approach takes the description of a programming task and provides a comprehensive solution for the task.
- *Pre-trained Word2API* (Li et al. 2018b): This approach is designed to use the pre-trained Word2API model and directly predict the related API methods.
- *Pre-trained DeepAPI* (Gu et al. 2016): This approach is designed to use the pre-trained DeepAPI model and directly predict the related API methods.

4.6.2 Model Training and Parameters Tuning

In this RQ, we use the Stack Overflow dataset to do the evaluation. Following the existing approach (e.g., *BIKER* and *CROKAGE*), we use question titles as input for all approaches, and the output should be the related API documents. *Word2API* and *DeepAPI* are trained on code and comment pairs. However, our datasets do not have code and comment information. We use the published pre-trained models to do the ranking. *Word2API* generates API methods with scores as output, and we directly use the scores to rank relevant API documents.

Given N questions, *DeepAPI* generates N API sequences for the questions; each sequence is for a question. We give the score for each sequence S_i to all API methods contained in the sequence S_i . If the API methods have multiple scores, we pick the highest one. If there is a tie during the ranking, we give them the same rank in the output.

As for *BIKER* and *CROKAGE*, we do a similar parameter tuning as in RQ2. For RAP4DQ, we follow the same process of tuning in RQ1.

4.7 RQ4. (Sensitivity) Analysis Approach.

We evaluate the impact of different factors on API forums.

Factor 1: Different API documentation weighting schemes. In Step 3 of RAP4DQ, we add a weight to each document. Here, we evaluate four weighting schemes: (1) no weight; (2) a trainable parameter, p_i ; (3) T_i / T_{max} (introduced in Section 3.3); and (4) $p_i (T_i / T_{max})$ (i.e., the one we use in RAP4DQ introduced in Section 3.3). For (2), it is a common way of adding weights in deep learning, while (3) is the way to calculate the frequency of usage in the dataset. The (4) is the combining of (2) and (3). For the weighting scheme (2), we argue that different documents should have different weights in the process, so we want to generate specialized weights for each document. For the weighting scheme (3), we aim to consider the normalized document frequency for each API document. For the weighting

scheme (4), we combine the weighting scheme (2) and (4) to generate more specialized weights for each API document. To test the impact of different weighting schemes, we set the other three factors as: Random Selection (Factor 2), ten negative samples (Factor 3), and with Learning-to-Rank (Factor 4).

Factor 2: Different algorithms for selecting negative samples. We test seven algorithms for selecting negative samples: (1) Random Selection; (2) TF-IDF + Cosine Similarity; (3) word2vec + Cosine Similarity, to select the irrelevant API documents as negative samples in RAP4DQ. The (1) is the most basic way that regarding all documentation as the same important and select one, the (2) are comparing the word frequency and documentation frequency, and the (3) are the word content similarity. The reason why we choose these three algorithms to do the testing is that we want to check if the word frequency, the documentation frequency, and the word content can help select the best negative samples. To test the impact of different algorithms for selecting negative samples, we set the other three factors as: $p_i(T_i/T_{max})$ (Factor 1), ten negative samples (Factor 3), and with Learning-to-Rank (Factor 4).

Factor 3: Different numbers of negative samples. We set the number of negative samples to 0, 1, 5, 10, 15, 20, 25, 30, or 35. To test the impact of different numbers of negative samples, we set the other three factors as: $p_i(T_i/T_{max})$ (Factor 1), Random Selection (Factor 2), and with Learning-to-Rank (Factor 4).

Factor 4: With/Without Learning-to-Rank. We test the performance of RAP4DQ with and without learning-to-rank. To test the impact of Learning-to-Rank, we set the other three factors as: $p_i(T_i/T_{max})$ (Factor 1), Random Selection (Factor 2), and ten negative samples (Factor 3).

Factor 5: Impact of Different Components of RAP4DQ. We evaluate the contribution of different Components by removing them one by one from the full model to see how much the removed feature can affect the performance of our approach on the API forum datasets. We remove the following key features in order: adding weights; using negative samples; learning to rank.

Factor 6: Impact of Different Data Splitting. We evaluate the influence of different data splitting on RAP4DQ. We test the following data splitting (training/tuning/testing): 80%/10%/10%, 70%/15%/15%, 60%/20%/20%, 50%/25%/25%.

Factor 7: Time Complexity. We collect the training and prediction time.

5 Experiment Results

5.1 Results for Unsupervised Question Answering Comparative Study on API Forums (RQ1)

Table 4 shows that on three API forum datasets, RAP4DQ outperforms 12 state-of-the-art unsupervised baselines on recommending relevant API documents to developer questions on all three API forums.

AUC and Accuracy metrics: RAP4DQ relatively improves the best-performed baseline GPT-2 on twitter and eBay datasets by 16.7% and 12.8% for AUC and the worst baseline TF-IDF on twitter and eBay dataset by 68.0% and 69.2% for AUC. RAP4DQ improves the best-performed baseline EMLO on AdWords dataset by 27.0% for AUC and the worst baseline TF-IDF on AdWords by 84.3%. It means that means the ability of RAP4DQ to classify the related API methods and the unrelated methods is better than all baselines. RAP4DQ also improves the best-performed baseline GPT-2 on twitter and eBay dataset

Table 4 RQ1 Results: Comparison with the unsupervised baselines in recommending relevant API documentation on three API forum datasets

Twitter	AUC	Accuracy	MAP	MRR	NDCG
TF-IDF	0.5	0.11	0.08	0.09	0.07
BM25	0.51	0.14	0.09	0.11	0.09
IDF+word2vec	0.71	0.46	0.14	0.13	0.12
CentIDF+KNN	0.72	0.49	0.18	0.19	0.19
CentIDF+ANN	0.72	0.49	0.18	0.19	0.19
CentIDF+KNN+WMD	0.63	0.37	0.23	0.26	0.21
CentIDF+ANN+WMD	0.63	0.37	0.23	0.26	0.21
WMD	0.7	0.50	0.31	0.33	0.33
ELMo (Pre-trained)	0.69	0.54	0.39	0.40	0.41
Bert (Pre-trained)	n/a	n/a	n/a	n/a	n/a
GPT-2 (Pre-trained)	0.72	0.57	0.43	0.45	0.45
RAP4DQ	0.84	0.61	0.46	0.48	0.55
eBay	AUC	Accuracy	MAP	MRR	NDCG
TF-IDF	0.52	0.09	0.07	0.08	0.06
BM25	0.52	0.10	0.08	0.09	0.07
Word2vec	0.72	0.23	0.07	0.11	0.07
IDF+word2vec	0.65	0.25	0.11	0.13	0.10
CentIDF+KNN	0.75	0.39	0.14	0.15	0.12
CentIDF+ANN	0.75	0.39	0.14	0.15	0.12
CentIDF+KNN+WMD	0.61	0.42	0.19	0.21	0.15
CentIDF+ANN+WMD	0.61	0.42	0.19	0.21	0.15
WMD	0.81	0.45	0.21	0.25	0.19
ELMo (Pre-trained)	0.72	0.51	0.25	0.33	0.28
Bert (Pre-trained)	n/a	n/a	n/a	n/a	n/a
GPT-2 (Pre-trained)	0.78	0.53	0.31	0.38	0.32
RAP4DQ	0.88	0.65	0.36	0.45	0.37
AdWords	AUC	Accuracy	MAP	MRR	NDCG
TF-IDF	0.51	0.11	0.06	0.09	0.11
BM25	0.51	0.14	0.08	0.11	0.12
Word2vec	0.69	0.15	0.10	0.13	0.15
IDF+word2vec	0.6	0.18	0.11	0.15	0.17
CentIDF+KNN	0.58	0.19	0.14	0.17	0.21
CentIDF+ANN	0.58	0.19	0.14	0.17	0.21
CentIDF+KNN+WMD	0.57	0.22	0.21	0.24	0.23
CentIDF+ANN+WMD	0.57	0.22	0.21	0.24	0.23
WMD	0.56	0.25	0.28	0.29	0.25
ELMo (Pre-trained)	0.74	0.31	0.33	0.36	0.32
Bert (Pre-trained)	n/a	n/a	n/a	n/a	n/a
GPT-2 (Pre-trained)	n/a	n/a	n/a	n/	n/a
RAP4DQ	0.94	0.44	0.42	0.48	0.41

for accuracy by 7.0% and 22.6% and the worst baseline TF-IDF by 454.5% and 622.2%. RAP4DQ improves the best-performed baseline ELMo on AdWords dataset for accuracy by 41.9% and the worst baseline TF-IDF by 300.0%. The higher accuracy means that more API methods we predict correctly. By considering both the AUC and accuracy, RAP4DQ shows better performance on the overall fitness.

Ranking-based metrics: RAP4DQ can improve the MAP, MRR, and NDCG up to 600.0%, 462.5%, and 685.7%, respectively. Higher MAP, MRR, NDCG indicates that RAP4DQ achieves the ranking closer to the perfect ranking and the related API methods appear higher in the top list.

Specifically, no individual approach consistently outperforms other unsupervised approaches. However, *CentIDF+KNN* performs better than others on eBay while *Word2vec* performs better than others on Twitter and AdWords. RAP4DQ can significant improve *CentIDF+KNN* and *Word2vec* by 16.7%, 17.3%, 62.1% and 15.1%, 22.2%, 36.2%, on Twitter, eBay, and AdWords for AUC, respectively. The pre-trained Bert cannot run on our dataset as its input length limit is around 500 words. However, some of the API methods' documentation and questions have more than 500 words. The pre-trained GPT-2 model cannot run on AdWords because it also has an input limit (around 4,000 words). In the AdWords dataset, there are several documents with 5,000-6,000 words, which is out of the input limit.

As for the results of RAP4DQ, it performs the best in dataset AdWords, and on the Twitter and eBay datasets, RAP4DQ performs a little bit lower. The reason for this is because, on the AdWords forum, the developers ask questions more specifically on the API usage, while on the eBay forum, the developers often ask questions more at a high level with fewer details. Without detailed information, our model is harder to find the key information for related API documentation. As for the Twitter forum, the developers' questions are much longer than on the other two forums. More words mean more biases for the model to catch the useful information, so that's the reason why RAP4DQ performs worst on the Twitter dataset.

5.2 Results for Supervised Question Answering Comparative Study on API Forums

Table 5 shows that RAP4DQ outperforms 4 state-of-the-art supervised baselines on recommending relevant API documents to developer questions on all three API forums.

AUC and Accuracy metrics: RAP4DQ relatively improves the best performed baseline GPT-2 on twitter and eBay dataset by 6.3% and 7.3% for AUC and the worst baseline BIKER on twitter and eBay dataset by 20.0% and 66.0% for AUC. RAP4DQ improves the best performed baseline EMLo on AdWords dataset by 17.5% for AUC and the worst baseline CROKAGE on AdWords by 59.3%. RAP4DQ also improves the best performed baseline BIKER on twitter and AdWords dataset for Accuracy by 5.2% and 4.8% and the worst baseline Seq2seq by 32.6% and 33.3%. RAP4DQ improves the best performed baseline GPT-2 on eBay dataset for Accuracy by 6.6% and the worst baseline CROKAGE by 160.0%.

Ranking based metrics: RAP4DQ can improve the MAP, MRR, and NDCG up to 250.0%, 220.0%, and 54.2% respectively. On twitter dataset, QDLinker, and RAP4DQ have the same NDCG score while on AdWords dataset, BIKER has a higher NDCG score than RAP4DQ. But considering all metrics, RAP4DQ is still the best performed approach.

Specifically, RAP4DQ and *GPT-2* perform better than others on Twitter and eBay forums, but the *GPT-2* cannot run on AdWords dataset. Comparing with RAP4DQ, RAP4DQ can improve *GPT-2* by 6.3% and 7.3% for AUC, on Twitter and eBay respectively. *ELMo* model performs better than all other baselines on the AdWords dataset, but our model still can

Table 5 RQ2 Results: Comparison with the supervised baselines in recommending relevant API documentation on three API forum datasets

Twitter	AUC	Accuracy	MAP	MRR	NDCG
Seq2seq	0.70	0.46	0.31	0.35	0.48
Seq2seq attention	0.71	0.48	0.35	0.34	0.51
QDLinker	0.76	0.51	0.37	0.39	0.55
ELMo (Re-trained)	0.78	0.55	0.39	0.38	0.50
Bert (Fine-tuned)	n/a	n/a	n/a	n/a	n/a
GPT-2 (Fine-tuned)	0.79	0.54	0.42	0.40	0.52
BIKER	0.68	0.58	0.41	0.41	0.50
CROKAGE	0.75	0.49	0.41	0.43	0.51
RAP4DQ	0.84	0.61	0.46	0.48	0.55
eBay	AUC	Accuracy	MAP	MRR	NDCG
Seq2seq	0.72	0.49	0.27	0.38	0.28
Seq2seq attention	0.74	0.48	0.29	0.41	0.24
QDLinker	0.79	0.52	0.32	0.40	0.33
ELMo (Re-trained)	0.80	0.55	0.31	0.42	0.32
Bert (Fine-tuned)	n/a	n/a	n/a	n/a	n/a
GPT-2 (Fine-tuned)	0.82	0.61	0.33	0.43	0.34
BIKER	0.53	0.36	0.20	0.24	0.31
CROKAGE	0.56	0.25	0.20	0.22	0.33
RAP4DQ	0.88	0.65	0.36	0.45	0.37
AdWords	AUC	Accuracy	MAP	MRR	NDCG
Seq2seq	0.66	0.33	0.31	0.38	0.33
Seq2seq attention	0.66	0.35	0.32	0.41	0.35
QDLinker	0.71	0.41	0.35	0.44	0.38
ELMo (Re-trained)	0.80	0.39	0.39	0.42	0.37
Bert (Fine-tuned)	n/a	n/a	n/a	n/a	n/a
GPT-2 (Fine-tuned)	n/a	n/a	n/a	n/a	n/a
BIKER	0.71	0.42	0.39	0.43	0.49
CROKAGE	0.59	0.34	0.11	0.13	0.24
RAP4DQ	0.94	0.44	0.42	0.48	0.41

improve the AUC by 17.5% on AdWords. The Bert still cannot run on our dataset based on the same reason mentioned in the RQ1.

5.3 Results for Supervised Question Answering Comparative Study on Stack Overflow. (RQ3)

Similar to in the API forum datasets in RQ1 and RQ2, in Table 6, it shows the running results for RAP4DQ and baselines on Stack Overflow dataset. The results in the table show that RAP4DQ can outperform all baselines on all metrics.

AUC and Accuracy metrics: RAP4DQ relatively improves the best-performed baseline DeepAPI by 17.1% for AUC and the worst baseline Word2API by 117.1% for AUC.

Table 6 RQ3 Results: Comparison with the java related question answering baselines in recommending relevant API documentation on stack overflow datasets

	AUC	Accuracy	MAP	MRR	NDCG
BIKER	0.73	0.19	0.41	0.44	0.52
CROKAGE	0.71	0.34	0.40	0.41	0.48
Word2API	0.41	0.45	0.18	0.20	0.26
DeepAPI	0.76	0.34	0.44	0.47	0.50
RAP4DQ	0.89 (↑ 17.1%)	0.85 (↑ 88.9%)	0.50 (↑ 11.1%)	0.53 (↑ 12.8%)	0.62 (↑ 24.0%)

↑ $XX\%$ indicates the improvements comparing with the best performed baseline

RAP4DQ also improves the best-performed baseline Word2API for Accuracy by 88.9% and the worst baseline BIKER by 347.4%.

Ranking-based metrics: RAP4DQ can improve the MAP, MRR, and NDCG up to 177.8%, 165.0%, and 138.5%, respectively. And comparing with the baseline best-performed results, RAP4DQ can improve the MAP, MRR, and NDCG at least for 11.1%, 12.8%, and 24.0%, respectively.

The results show that even on the OS API dataset, compared with existing OS API related question answering approaches, RAP4DQ can perform better on all evaluation metrics which proves that even though our approach is designed for web API question answering, for OS API related question answering, RAP4DQ is still useful and reliable.

5.4 Results for Sensitivity Analysis (RQ4)

Table 7 shows that **different API documentation weighting schemes can affect the AUC of RAP4DQ**. Adding weights to differentiate API documents can improve the results of RAP4DQ, and $p_i(T_i/T_{max})$ achieves the best results compared with the other three weighting schemes. This result proves that adding weights for different API documentation is helpful. Also, only adding a trainable weight or only considering the popularity of the documentation is not enough. Combining these two together can help the model to figure out the importance of different API documentation the most.

Table 8 shows that **different algorithms for selecting negative samples have an impact on RAP4DQ**. Overall, Random Selection can outperform the other two algorithms. Therefore, we use Random Selection in RAP4DQ. Our explanation is that Random Selection may bring high diversities in negative samples. For example, from Fig. 4, we can see the cosine similarity between selected negative samples and the related documents. The figure shows that the negative samples selected by the Random Selection are more diversified, while the negative samples selected by the other two weighting schemes selected have cosine similarity scores close to 1.

Table 7 RQ4. Results of Factor 1: Different API documentation weighting schemes

	Category	Twitter	eBay	AdWords
	No Weight	0.79	0.82	0.87
	A trainable parameter, p_i	0.83	0.84	0.87
	T_i/T_{max}	0.81	0.86	0.89
	$p_i(T_i/T_{max})$	0.84	0.88	0.94

Table 8 RQ4. Results of Factor 2: Different algorithms for selecting negative samples

Category	Twitter	eBay	AdWords
Random Selection	0.84	0.88	0.94
TF-IDF + CS	0.76	0.82	0.86
Word2vec + CS	0.79	0.83	0.88

CS: Cosine Similarity; Training setting: $p_i(T_i/T_{max})$ (Factor 1), ten negative samples (Factor 3), and with Learning-to-Rank (Factor 4)

Table 9 shows that **the number of negative samples can affect the results of RAP4DQ**. Adding more negative samples does not improve the results. On the contrary, it can hurt the results. And if the number of negative samples is not enough, the model cannot separate the most suitable documentation from other similar documentation. So as a conclusion, on our dataset, selecting ten negative samples can help us achieve the best results.

Table 10 shows that **the learning-to-rank approach can help increase the accuracy of RAP4DQ**. Our idea of using the learning-to-rank to rank the API documentation has been proved to be useful and can improve the performance of the overall listing. This result also shows that our idea of separating similar documentation is important, and only adding weights on different documentation is not enough to achieve this key point.

Table 11 shows that by removing each component from the model, the AUC on three API forum datasets will decrease. It proves that each component of our model can contribute to the final performance of the model. To be more specific, by removing wights, the AUC decrease 2.4%, 2.3%, 5.6% on twitter, eBay, and AdWords. When removing negative samples, the AUC decrease 2.5%, 4.9%, 4.7% on twitter, eBay, and AdWords. And when removing learning to rank, the AUC decrease 11.1%, 9.3%, 11.8% on twitter, eBay, and AdWords. The results show that learning to rank influences the results the most.

Table 12 shows that if the training dataset size is over 60% of overall data, the performances of RAP4DQ are comparable. Comparing the settings of 80%/10%/10% and 60%/20%/20%, the training data decreased 20%. However, the performance in terms of AUC was only reduced by 4%-5% on three API forums. Furthermore, comparing between

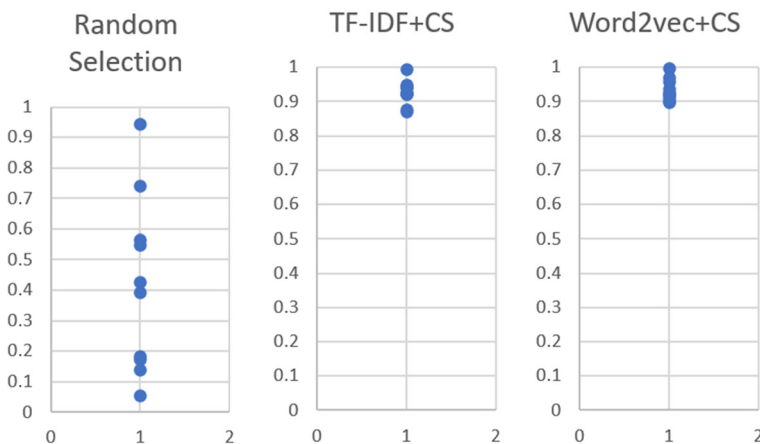


Fig. 4 RQ4. Cosine Similarity Distribution for Different Weighting Schemes

Table 9 RQ4. Results of Factor 3: Different numbers of selected negative samples

# of Negative Samples	Twitter	eBay	AdWords
0	0.74	0.76	0.79
1	0.78	0.82	0.83
5	0.81	0.84	0.87
10	0.84	0.88	0.94
15	0.83	0.87	0.91
20	0.79	0.88	0.86
25	0.75	0.84	0.82
30	0.74	0.79	0.76
35	0.72	0.74	0.74

Training setting: $p_i(T_i/T_{max})$ (Factor 1), Random Selection (Factor 2), and with Learning-to-Rank (Factor 4)

the settings of 80%/10%/10% and 50%/25%/25%, the training data decreased 30%. However, the performance in terms of AUC was only reduced by 11%-13% on three API forums.

Table 13 shows the time cost for running our approach and supervised baselines on three API forum datasets. On three API forum datasets, our model all takes about 6 hours to do the training and 2-3 seconds for the prediction. Deep learning-based approaches are known to take more time for training. However, for prediction, the speed of RAP4DQ is at the same level as the best-performed baselines, and our approach performs better.

6 Discussion and Implication

Let us present one in-depth case studies to understand why RAP4DQ can work well and why it can perform better than the baselines.

Case Study Figure 5 shows an unclear question with only word description on eBay. The developer wanted to know how to find an item by using API functions with some specific information. The eBay API development team suggested the questioner see the API documentation for *findItemsAdvance*.

To analyze why our approach can pick the correct API method *findItems*—*Advance* for the question, we select a base model without adding weights, using negative samples, and using learning to rank these three components and test the performance change by adding each component one by one. From Fig. 5b, we can see that if we use the base model, the model can get the correct API method on rank nine which is the performance for the basic GRU model. And then, if we add learning to rank to the base model, the reranking can help us pick the correct API method at the fifth position in the ranking list. Similarly, if we add

Table 10 RQ4. Results of Factor 4: With/without learning-to-rank

	Twitter	eBay	AdWords
Without Learning-to-Rank	0.78	0.85	0.77
With Learning-to-Rank	0.84 (↑ 7.7%)	0.88 (↑ 10.6%)	0.94 (↑ 22.1%)

↑ $XX\%$ indicates the improvements; Training setting: $p_i(T_i/T_{max})$ (Factor 1), Random Selection (Factor 2), and ten negative samples (Factor 3)

Table 11 RQ4. Results of Factor 5: Impact of key different components of RAP4DQ

	Twitter	eBay	AdWords
RAP4DQ	0.84	0.88	0.94
w/o weights	0.82	0.86	0.89
w/o weights and negative samples	0.80	0.82	0.85
w/o weights, negative samples and learning to rank	0.72	0.75	0.76

negative samples, the model can predict the correct API method at the third position. Combining all the components, we can see that each key component could improve the accuracy of the model prediction. By using all of them, our approach can have higher accuracy in predicting the correct API methods for the questions.

Approach Limitations. Through the manual analysis of the results, we identify the following limitations of RAP4DQ:

First, RAP4DQ *does not work well on the questions that contain too much code and few words*. In RAP4DQ, we only analyze the tokens to catch information. This kind of question is hard for our model to deal with. In future work, we are planning to do a more detailed code analysis that can help improve the performance of our model on this. Second, RAP4DQ *does not work well on the questions that are too general*. If a question is too general, it is even hard for the API development team to give a proper recommendation.

7 Threats to Validity

We identify the following threats to validity:

Implementation of baselines To compare with existing approaches, we directly use the published code of the baselines, except *QDLinker* (Li et al. 2018a). The code of *QDLinker* is not publicly available. We completely implemented *QDLinker*. The *QDLinker* paper reported slightly higher results than what we reported using our implementation of *QDLinker* in this paper. One possible reason is that *QDLinker* performs differently on varied datasets, and some implementation details are not mentioned in their paper, which may make our version of *QDLinker* slightly different from the one in the original paper. However, we tried our best to build and tune the *QDLinker* parameters on our dataset, and this is the best effort we can make when the code is not publicly available. We tuned RAP4DQ and *QDLinker* both on our dataset, which makes it fair for both RAP4DQ and *QDLinker*. Also, the link for the baseline *CROKAGE* is not working. So similar as *QDLinker*, we also completely implemented *CROKAGE* and evaluated it with other approaches fairly just like *QDLinker*.

Table 12 RQ4. Results of Factor 6: Impact of different data splitting

Training/Tuning/Testing	Twitter	eBay	AdWords
80%/10%/10%	0.84	0.88	0.94
70%/15%/15%	0.82	0.87	0.91
60%/20%/20%	0.79	0.84	0.89
50%/25%/25%	0.73	0.76	0.81

Table 13 RQ4. Results of Factor 7: Time complexity

	Training (Minutes)			Prediction (Seconds)		
	Twitter	eBay	AdWords	Twitter	eBay	AdWords
Seq2seq	251	243	264	1	1	1
Seq2seq attention	273	299	287	1	1	1
QDLinker	169	175	172	1	1	1
ELMo (Re-trained)	1048	991	1217	14	13	17
Bert (Fine-tuned)	n/a	n/a	n/a	n/a	n/a	n/a
GPT-2 (Fine-tuned)	416	397	n/a	17	21	n/a
BIKER	1	1	1	1	1	1
CROKAGE	1	1	1	1	1	1
RAP4DQ	372	351	395	2	2	3

devendrkuma_203 asked · Mar 28 '17 at 5:14 PM

I need product listing, pagination, filter by category, country, quantity sold, using xml api in my php site

hi, I need product listing, pagination, filter by category, country, quantity sold I am not getting which api is suitable for this, because some api not gives details which I need, if any other api gives that detail but they not provide pagination and filter. Help me out to sort out this please.

xml response

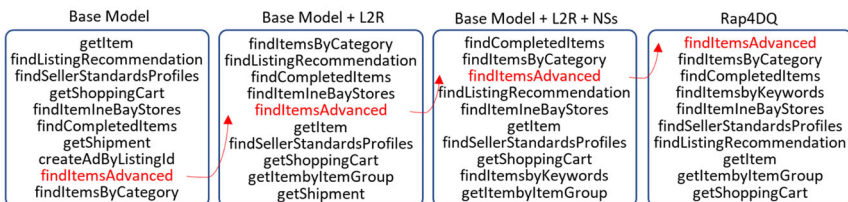
Comment

1 Answer · Write an Answer

catherine_developersupport answered · Mar 29 '17 at 5:22 PM

@devendrkuma_203, please try [findItemsAdvanced][1] [1]: <http://developer.ebay.com/Devzone/finding/CallRef/findItemsAdvanced.html>

(a) A Question with the Answer from the eBay Development API Team.



(b) Result Changes by Adding Key Features. L2N: Learning to Rank; NSs: Negative Samples

Fig. 5 Case Study

Manual construction of labeled golden set To evaluate approaches, it is necessary to have a labeled golden set. During the labeling process, most answers can clearly show the relevant APIs. However, some answers can contain outdated links for API documents, which makes it very difficult to determine the right relevant APIs. We discard such answers to try our best to minimize the bias. Although this part of work is common in question and answering, this process will bring bias to our results since the authors of this paper are not from API teams.

Selection of API forums There are many API forums for different APIs. In our research, we only focused on three very popular API forums, Twitter, eBay, and AdWords. Thus, we cannot claim that RAP4DQ is generic for all API forums. However, the key drivers of RAP4DQ outperforming the baselines are general across forums.

8 Related Work

Here, we summarize the related work to our study.

Statistical and Language Model Based Question and Answer Retrieval Recently, extensive research using statistical and language models has been proposed to improve question and answer retrieval (Cao et al. 2010; Figueroa and Neumann 2016; Jeon et al. 2005; Duan et al. 2008; Ji et al. 2012; Zhou et al. 2011; Sakai et al. 2011; Sun et al. 2005; Surdeanu et al. 2008; Yen et al. 2013; Xue et al. 2008; Berger et al. 2000; Yao et al. 2015; Nicosia et al. 2015; Singh and Simperl 2016; Tan et al. 2016; Burke et al. 1997; Zhou et al. 2013; Brokos et al. 2016). For example, Yen et al. (2013) propose a model to categorize the answer type of a given question and build a context-ranking model to re-rank retrieved documentation. Burke et al. (1997) and Zhou et al. (2013) use WordNet and Wikipedia to expand the semantics of questions to improve the question retrieval. Sakai et al. (2011) propose an approach to build an answer selection system involving multiple answer assessors and graded-relevance information retrieval metrics. Yao et al. (2015) report a set of rules to detect high-quality questions and help users to identify a useful answer that can gain positive feedback from other users. Sun et al. (2005) study the dependency relationships among the matched terms of documentation and questions, and use the analyzed relationships to rank answers. Nicosia et al. (2015) propose an automatic answer selection system to detect the right answer for the target questions by incorporating the position of answers in question threads. Singh and Simperl (2016) present a searching system using semantic keyword search to find similar questions between answered questions and unanswered ones. Tan et al. (2016) develop machine learning models to find different sections in answers that are suitable to describe the complex semantic relationships between questions and their answers. Brokos et al. (2016) proposed the ways of improving traditional answer retrieval solutions with Word Mover Distance (Kusner et al. 2015) in order to improve question and answer retrieval techniques have higher results. Word Mover Distance (Kusner et al. 2015) is a statistical model that measures the dissimilarity between two text documents as the minimum amount of distance that the embedded words of one document need to "Travel" to reach the embedded words of another document. None of the above-mentioned approaches are proposed for developer Q&A websites. However, we borrowed some techniques to implement our baselines, such as Word Mover Distance (Kusner et al. 2015), in this paper. Through our experiments, RAP4DQ can outperform the baselines.

Deep Learning Based Question and Answer Retrieval In recent years, deep learning has become popular in question and answer retrieval. Some deep learning based approaches have been proposed (Er et al. 2016; Kokkinos and Margaritis 2015; Nassif et al. 2016; Severyn and Moschitti 2015, 2016; Zhou et al. 2015, 2016; Yan et al. 2016; Li et al. 2018a; Palangi et al. 2016; Guo et al. 2016). For example, Severyn and Moschitti (2015) propose a convolution neural network (CNN) (Kim 2014) architecture to embed short-text questions and documentation to their distributed vectors for effective retrieval. Yan et al. (Yan et al. 2016) learn a deep neural network (DNN) to detect the relationships between the query context and the answers. Palangi et al. (2016) develop a model with LSTM to improve answer selection. Nassif et al. (2016) design a model based on stacked bidirectional Long Short-term Memory (LSTM)s (Hochreiter and Schmidhuber 1997) and Multi-layer Perceptron (MLP) (Pal and Mitra 1992) to search the relationships on semantics between questions and answers. Zhou et al. (2015, 2016) used a deep neural network to learn the semantic of questions and answers to solve the mismatch question problem. However, unlike the above-mentioned studies, we aim to directly link questions with API documentation without retrieving answers.

Also, answering the open-domain questions is also a popular topic in recent years (He et al. 2020; Cao et al. 2020). He et al. (2020) propose a model using BERT or other state-of-the-art contextual language models to learn representations from QA data to answer open-domain questions. Cao et al. (2020) build a decomposed transformer, which substitutes the full self-attention with question-wide and passage-wide self-attentions in the lower layers. But as we mentioned in the introduction section, these open-domain approaches can only deal with short questions with short answers while RAP4DQ can deal with. And also, for open-domain Q&A, they often need to catch the relationship between documentation to find the most suitable answer, while in our problem, we often do not the order and the relationship between documentation which makes the open-domain Q&A approaches cannot work. For example, within the example in Fig. 3, there are two relate API methods `Webpage` and `WebpageParameter`. From the developer team's answer, we don't know the order of these two API methods, and this order does not influence the developer to solve this problem after carefully reading the documentation for this two API documentation. But for open-domain Q&A approaches, they cannot build the relationship network to analyze and find the best results without knowing the order of API methods in the answers. That's also the difference between our problem and open-domain question answering.

Some existing approaches (Huang et al. 2018; Li et al. 2018a, b; Silva et al. 2019; Gu et al. 2016) are similar to our approach. BIKER (Huang et al. 2018) uses similar question retrieval to pick relevant API methods. Word2API (Li et al. 2018b) develop a model to learn the relatedness of words and APIs. CROKAGE (Silva et al. 2019) presents a searching system to search relevant code examples on the web for programming tasks. DeepAPI (Gu et al. 2016) construct an RNN encoder-decoder model to generate API usage sequences for a given query. All these four existing studies are designed and evaluated on the Stack Overflow data for Java APIS, while our approach is mainly designed for API forums and web APIs. And the most similar work to ours is QDLinker (Li et al. 2018a). QDLinker utilizes a deep neural network to retrieve the top 50 relevant Java API documentation for a developer question and use a learning-to-rank technique to re-rank the retrieved 50 Java API documents. The major differences between RAP4DQ and QDLinker are listed as follows: First, RAP4DQ is designed to train on positive and negative samples to add more discriminative power to distinguish API documentation with similar descriptions. Second, RAP4DQ differentiates API documentation by adding weights to various API documentation. However,

QDLinker treats all API documentation equally. Third, QDLinker is proposed for Stack Overflow and only Java documentation. However, RAP4DQ is developed for API Q&A forums and three web API documentation.

9 Conclusion

Directly recommending relevant API documentation to answer a developer question on API Q&A forums can be very useful. However, little research has been done on helping answer developer questions on API Q&A forums. In this paper, we propose a deep learning-based approach that identifies relevant API documentation from the API documentation list to answer a developer question. Specifically, RAP4DQ employs word2vec to obtain word embedding vectors for the words in API documentation and questions. Second, we design RAP4DQ to train on positive and negative samples. Third, we use two GRUs to train question and API documentation embedding vectors separately and differentiate distinct API documents with different importances by having the weights. Last, we design a new loss function to effectively train RAP4DQ to distinguish relevant API documentation from irrelevant documentation.

We evaluate RAP4DQ against a set of state-of-the-art baselines on three web API forums and the Stack Overflow. The empirical results show that RAP4DQ can outperform all of the studied state-of-the-art approaches on all three studied API forums and the Stack Overflow. Specifically, we can gain a relative improvement up to 84.3% in terms of AUC score on API forums and 117.1% in terms of AUC score on the Stack Overflow.

We plan to study more API forums and create a larger labeled dataset. We also plan to do word-level similarity analysis instead of only on the whole question/documentation level. Moreover, we will test RAP4DQ on Stack Overflow data. Through the analysis, we find that API documentation and questions can contain a lot of code and sometimes incomplete code blocks. Therefore, we plan to improve our code processing and modeling.

Acknowledgements We thank the anonymous reviewers who reviewed our paper and the associated editor for their valuable feedback.

References

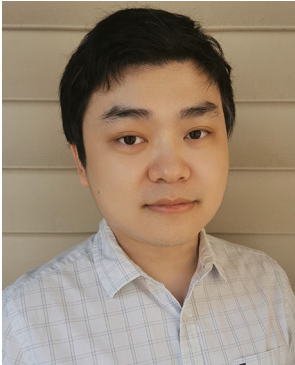
- AdWords (2019) A adwords question: How to create dynamic targeting 'all websites'. <https://groups.google.com/forum/#!topic/adwords-api/xPIhAyhAX9o>. Last Accessed May 10, 2019
- Adwords (2020) Adwords. <https://ads.google.com/>
- Annoy (2020) Annoy. URL <https://github.com/spotify/annoy>
- Berger A, Caruana R, Cohn D, Freitag D, Mittal V (2000) Bridging the lexical chasm: statistical approaches to answer-finding. In: Proceedings of the 23rd annual international ACM SIGIR conference on Research and development in information retrieval. ACM, pp 192–199
- Bishop CM (2006) Pattern recognition and machine learning. Springer
- Brokos G-I, Malakasiotis P, Androutsopoulos I (2016) Using centroids of word embeddings and word mover's distance for biomedical document retrieval in question answering. arXiv:1608.03905
- Burke RD, Hammond KJ, Kulyukin V, Lytinen SL, Tomuro N, Schoenberg S (1997) Question answering from frequently asked question files: Experiences with the faq finder system. *AI Mag* 18(2):57–57
- Cao Q, Trivedi H, Balasubramanian A, Balasubramanian N (2020) Deformer: Decomposing pre-trained transformers for faster question answering. arXiv:2005.00697
- Cao X, Cong G, Cui B, Jensen CS (2010) A generalized framework of exploring category information for question retrieval in community question answer archives. In: Proceedings of the 19th international conference on World wide web. ACM, pp 201–210

- Cho K, Van Merriënboer B, Gulcehre C, Bahdanau D, Bougares F, Schwenk H, Bengio Y (2014) Learning phrase representations using rnn encoder-decoder for statistical machine translation. arXiv:1406.1078
- Devlin J, Chang M-W, Lee K, Toutanova K (2018) Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv:1810.04805
- Duan H, Cao Y, Lin C-Y, Yu Y (2008) Searching questions by identifying question topic and question focus. Proceedings of ACL-08: HLT, pp 156–164
- Ebay (2019) A ebay question: How to find product descriptions by id? <https://forums.developer.ebay.com/questions/16455/how-to-find-product-descriptions-by-id.html>. Last Accessed May 10, 2019
- eBay (2020) ebay. <https://www.ebay.com/>
- Er MJ, Zhang Y, Wang N, Pratama M (2016) Attention pooling-based convolutional neural network for sentence modelling. Inf Sci 373:388–403
- Figuerola A, Neumann G (2016) Context-aware semantic classification of search queries for browsing community question–answering archives. Knowl-Based Syst 96:1–13
- Gu X, Zhang H, Zhang D, Kim S (2016) Deep api learning. In: Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering, pp 631–642
- Guo J, Fan Y, Ai Q, Bruce Croft W (2016) A deep relevance matching model for ad-hoc retrieval. In: Proceedings of the 25th ACM International on Conference on Information and Knowledge Management, pp 55–64
- He H, Ning Q, Roth D (2020) Quase: Question-answer driven sentence encoding. In: Proc. of the annual meeting of the association for computational linguistics (ACL)
- Hochreiter S, Schmidhuber J (1997) Long short-term memory. Neural Comput 9(8):1735–1780
- Huang Q, Xia X, Xing Z, Lo D, Wang X (2018) Api method recommendation without worrying about the task-api knowledge gap. In: 2018 33Rd IEEE/ACM international conference on automated software engineering (ASE). IEEE, pp 293–304
- Jeon J, Bruce Croft W, Lee JH (2005) Finding similar questions in large question and answer archives. In: Proceedings of the 14th ACM international conference on Information and knowledge management. ACM, pp 84–90
- Ji Z, Xu F, Wang B, He B (2012) Question-answer topic model for question retrieval in community question answering. In: Proceedings of the 21st ACM international conference on Information and knowledge management. ACM, pp 2471–2474
- Keras (2019) Keras documentation. <https://keras.io/>. Last Accessed May 10, 2019
- Kim Y (2014) Convolutional neural networks for sentence classification. arXiv:1408.5882
- Kokkinos Y, Margaritis KG (2015) Topology and simulations of a hierarchical markovian radial basis function neural network classifier. Inf Sci 294:612–627
- Kusner M, Yu S, Kolkin N, Weinberger K (2015) From word embeddings to document distances. In: International conference on machine learning, pp 957–966
- Li J, Sun A, Xing Z (2018a) Learning to answer programming questions with software documentation through social context embedding. Inf Sci 448:36–52
- Li X, Jiang H, Kamei Y, Chen X (2018b) Bridging semantic gaps between natural languages and apis with word embedding. IEEE Trans Softw Eng 46(10):1081–1097
- Li Y, Wang S, Nguyen TN (2020) An empirical study on the characteristics of question-answering process on developer forums. In: Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: Companion Proceedings, pp 318–319
- Lilleberg J, Zhu Y, Zhang Y (2015) Support vector machines and word2vec for text classification with semantic features. In: 2015 IEEE 14th international conference on cognitive informatics & cognitive computing (ICCI* CC). IEEE, pp 136–140
- Luong M-T (2015) Hieu pham, and christopher d manning. Effective approaches to attention-based neural machine translation. arXiv:1508.04025
- Mamykina L, Manoim B, Mittal M, Hripcsak G, Hartmann B (2011) Design lessons from the fastest q&a site in the west. In: Proceedings of the SIGCHI conference on Human factors in computing systems. ACM, pp 2857–2866
- Mikolov T, Chen K, Corrado G, Dean J (2013) Efficient estimation of word representations in vector space. arXiv:1301.3781
- Nassif H, Mohtarami M, Glass J (2016) Learning semantic relatedness in community question answering using neural models. In: Proceedings of the 1st Workshop on Representation Learning for NLP, pp 137–147
- Nicosia M, Filice S, Barrón-Cedeno A, Saleh I, Mubarak H, Gao W, Nakov P, Da San Martino G, Moschitti A, Darwish K et al (2015) Qcri: Answer selection for community question answering-experiments for arabic and english. In: Proceedings of the 9th International Workshop on Semantic Evaluation (SemEval 2015), pp 203–209

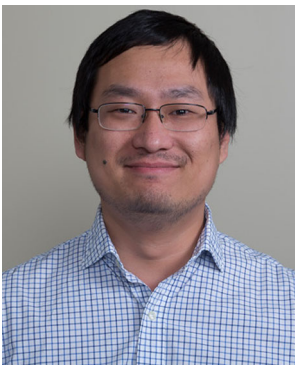
- NLTK (2020) Nltk. <https://www.nltk.org/>
- Pal SK, Mitra S (1992) Multilayer perceptron, fuzzy sets, and classification. *IEEE Trans Neural Netw* 3(5):683–697
- Palangi H, Li D, Shen Y, Gao J, He X, Chen J, Song X, Ward R (2016) Deep sentence embedding using long short-term memory networks: Analysis and application to information retrieval. *IEEE/ACM Trans Audio Speech Lang Process (TASLP)* 24(4):694–707
- Peters ME, Neumann M, Iyyer M, Gardner M, Clark C, Lee K, Zettlemoyer L (2018) Deep contextualized word representations. arXiv:1802.05365
- Radford A, Wu J, Child R, Luan D, Amodei D, Sutskever I (2019) Language models are unsupervised multitask learners. *OpenAI blog* 1(8):9
- Rahman MM, Roy C (2018) Effective reformulation of query for code search using crowdsourced knowledge and extra-large data analytics. In: 2018 IEEE International conference on software maintenance and evolution (ICSME). IEEE, pp 473–484
- Rahman MM, Roy CK, Lo D (2016) Rack: Automatic api recommendation using crowdsourced knowledge. In: 2016 IEEE 23rd international conference on software analysis, evolution, and reengineering (SANER), vol 1. IEEE, pp 349–359
- Rajaraman A, Ullman JD (2011) Mining of massive datasets. Cambridge University Press
- Ranklib (2020) Ranklib. <https://github.com/codelibs/ranklib>. Last Accessed Dec 9, 2020
- Rap4DQ Replication (2020) Rap4dq-replication. <https://github.com/spacenjiti/QA2020>
- Robertson S, Zaragoza H et al (2009) The probabilistic relevance framework: Bm25 and beyond. *Found Trends® Inf Retr* 3(4):333–389
- Sakai T, Ishikawa D, Kando N, Seki Y, Kuriyama K, Lin C-Y (2011) Using graded-relevance metrics for evaluating community qa answer selection. In: Proceedings of the fourth ACM international conference on Web search and data mining, pp 187–196. ACM
- Scikit-learn (2020) Scikit-learn. <https://scikit-learn.org/stable/>
- Severyn A, Moschitti A (2015) Learning to rank short text pairs with convolutional deep neural networks. In: Proceedings of the 38th international ACM SIGIR conference on research and development in information retrieval. ACM, pp 373–382
- Severyn A, Moschitti A (2016) Modeling relational information in question-answer pairs with convolutional neural networks. arXiv:1604.01178
- Silva RFG, Roy CK, Rahman MM, Schneider A, Paixao K, de Almeida Maia M (2019) Recommending comprehensive solutions for programming tasks by mining crowd knowledge. In: 2019 IEEE/ACM 27th international conference on program comprehension (ICPC). IEEE, pp 358–368
- Singh P, Simperl E (2016) Using semantics to search answers for unanswered questions in q&a forums. In: Proceedings of the 25th International Conference Companion on World Wide Web. International World Wide Web Conferences Steering Committee, pp 699–706
- Squire M (2015) "Should we move to stack overflow?" measuring the utility of social media for developer support. In: 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering, vol 2. IEEE, pp 219–228
- StackExchangeNetwork (2020) Stack overflow. <https://stackoverflow.com/>
- Sun R, Cui H, Li K, Kan M-Y, Chua T-S (2005) Dependency relation matching for answer selection. In: Proceedings of the 28th annual international ACM SIGIR conference on Research and development in information retrieval. ACM, pp 651–652
- Surdeanu M, Ciaramita M, Zaragoza H (2008) Learning to rank answers on large online qa collections. In: Proceedings of ACL-08: HLT, pp 719–727
- Sutskever I, Vinyals O, Le Quoc V (2014) Sequence to sequence learning with neural networks. In: Advances in neural information processing systems, pp 3104–3112
- Tan Mx, Santos CD, Xiang B, Zhou B (2016) Improved representation learning for question answer matching. In: Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), vol 1, pp 464–473
- Twitter (2020) Twitter. URL <https://twitter.com/>
- Uddin G, Khomh F (2017) Automatic summarization of api reviews. In: 2017 32nd IEEE/ACM international conference on automated software engineering (ASE). IEEE, pp 159–170
- Venkatesh PK, Wang S, Zhang F, Zou Y, Hassan AE (2016) What do client developers concern when using web apis? an empirical study on developer forums and stack overflow. In: 2016 IEEE International conference on web services (ICWS). IEEE, pp 131–138
- Wang S, Chen T-HP, Hassan AE (2018) How do users revise answers on technical q&a websites? a case study on stack overflow. *IEEE Transactions on Software Engineering*
- Wu Q, Burges CJC, Svore KM, Gao J (2010) Adapting boosting for information retrieval measures. *Inf Retr* 13(3):254–270

- Xue X, Jeon J, Bruce Croft W (2008) Retrieval models for question and answer archives. In: Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval. ACM, pp 475–482
- Yan R, Song Y, Wu H (2016) Learning to respond with deep neural networks for retrieval-based human-computer conversation system. In: Proceedings of the 39th International ACM SIGIR conference on Research and Development in Information Retrieval. ACM, pp 55–64
- Yao Y, Tong H, Xie T, Akoglu L, Xu F, Lu J (2015) Detecting high-quality posts in community question answering sites. *Inf Sci* 302:70–82
- Yen S-J, Wu Y-C, Yang J-C, Lee Y-S, Lee C-J, Liu J-J (2013) A support vector machine-based context-ranking model for question answering. *Inf Sci* 224:77–87
- Zhou G, Li C, Zhao J, Liu K (2011) Phrase-based translation model for question retrieval in community question answer archives. In: Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1. Association for Computational Linguistics, pp 653–662
- Zhou G, Liu Y, Liu F, Zeng D, Zhao J (2013) Improving question retrieval in community question answering using world knowledge. In: Twenty-third international joint conference on artificial intelligence
- Zhou G, He T, Zhao J, Hu P (2015) Learning continuous word embedding with metadata for question retrieval in community question answering. In: Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers), vol 1, pp 250–259
- Zhou G, Zhou Y, He T, Wu W (2016) Learning semantic representation with neural networks for community question answering retrieval. *Knowl-Based Syst* 93:75–83

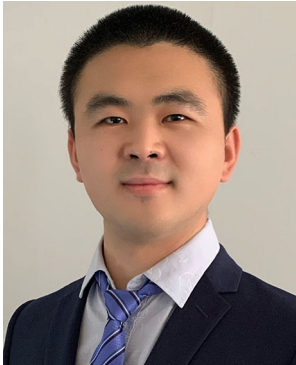
Publisher's note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.



Yi Li is a Ph.D. candidate in the Department of Informatics, Ying Wu College of Computing, New Jersey Institute of Technology. His research areas include software engineering research based on data science, systems, and artificial intelligence. He has published several papers in the OOPSLA, ASE, ICSE, FSE, MSR conferences, and the EMSE and TGIS journals.



Shaohua Wang is an Assistant Professor in the Department of Informatics at New Jersey Institute of Technology. His research lies the intersection of Software Engineering, Program Analysis, and Artificial Intelligence. He aims to build AI-based solutions to improve software quality, reliability, and intelligence. Particularly, his current research interests include, but not limited to, bug detection, automated program repair, testing, vulnerability detection, and interpretable and explainable models. More about Shaohua and his work is available online: <https://davidshaohuawang.wordpress.com/>.



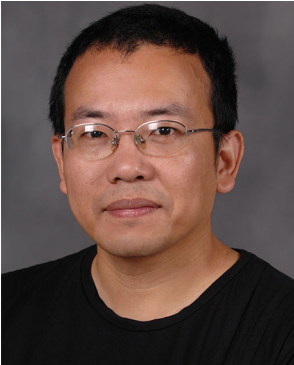
Wenbo Wang is a Ph.D. student in the Department of Informatics, College of Computing, New Jersey Institute of Technology. His research interests include Natural Language Processing, Machine Learning, Vulnerability Detection, etc. He has some publications on TGIS.



Tien N. Nguyen is currently a Full Professor in the Computer Science Department at the University of Texas at Dallas (UTD). His research interests include program analysis, software security, machine learning for software engineering and programming languages, software mining, and software maintenance and evolution. He has served as the Program Co-Chair of the 32nd ACM/IEEE International Conference on Automated Software Engineering (ASE 2017), and 4 times as the Chair of the Formal Demonstration Tracks at ICSE and FSE.




Yan Wang is an associate professor at the School of Information in Central University of Finance and Economics in China. He went on to study computer science at University of Windsor in Canada and received his Ph.D and master degrees respectively. His research interests are the areas of Deep Web, Open-domain conversation and Software Engineering.



Xinyue Ye is a Harold Adams Endowed Associate Professor at the Department of Landscape Architecture and Urban Planning at Texas A&M University. He holds a Ph.D. degree in Geographic Information Science from Joint Program between University of California at Santa Barbara and San Diego State University, a M.S. in Geographic Information Systems from Eastern Michigan University, and a M.A. in Hyman Geography from University of Wisconsin at Milwaukee. His research focuses on geospatial artificial intelligence, smart cities, spatial econometrics and urban computing.

Affiliations

Yi Li¹ · Shaohua Wang¹  · Wenbo Wang¹ · Tien N. Nguyen² · Yan Wang³ · Xinyue Ye⁴

Yi Li
yl622@njit.edu

Wenbo Wang
ww6@njit.edu

Tien N. Nguyen
Tien.N.Nguyen@utdallas.edu

Yan Wang
dayanking@gmail.com

Xinyue Ye
xinyue.ye@tamu.edu

¹ New Jersey Institute of Technology, University Heights, Newark, NJ 07102 USA

² The University of Texas at Dallas, 800 W. Campbell Road, Richardson, TX 75080-3021, USA

³ Central University of Finance and Economics, Changping District, Beijing, 100081, China

⁴ Texas A, M University, 400 Bizzell St, College Station, TX 77843, USA