DOI: 10.1111/tgis.12798

RESEARCH ARTICLE

Transactions in GIS WILEY

QA4GIS: A novel approach learning to answer GIS developer questions with API documentation

Wenbo Wang[1] | Yi Li[1] | Shaohua Wang[1] | Xinyue Ye[2]

[1]Department of Informatics, New Jersey Institute of Technology, University Heights, Newark, NJ, USA

[2]Department of Landscape Architecture and Urban Planning, Texas A&M University, College Station, TX, USA

Correspondence
Xinyue Ye, Department of Landscape Architecture and Urban Planning, Texas A&M University, College Station, TX 77840, USA.
Email: xinyue.ye@tamu.edu

Shaohua Wang, Department of Informatics, Ying Wu College of Computing, New Jersey Institute of Technology, Newark, NJ, USA
Email: davidsw@njit.edu

Funding information
National Science Foundation, Grant/Award Number: 1937908

Abstract

Community-based question answering websites have attracted more and more scholars and developers to discuss domain knowledge and software development. In this article, we focus on the GIS section of the Stack Exchange website and develop a novel approach, QA4GIS, a deep learning-based system for question answering tasks with a deep neural network (DNN) model to extract the representation of the query–API document pair. We use the LambdaMART model to rerank the candidate API documents. We begin with an empirical analysis of the questions and answers, demonstrating that API documents could answer 52.93% of the questions. Then we evaluate QA4GIS by comparing it with 10 other baselines. The experiment results show that QA4GIS can improve 21.39% on the MAP score and 22.34% on the MRR score compared with the best baseline SIF.

1 | INTRODUCTION

A geographic information system (GIS) is designed to store, process, visualize, and model spatial data. Both commercial and free/open-source GIS software packages have been widely used in many domains to promote spatial thinking and action. QGIS (previously known as Quantum GIS) is an open-source cross-platform software program. It is also a standout among the best-known open-source GIS programming frameworks. Furthermore, it is the most popular framework on the GIS section of the Stack Exchange (https://stackexchange.com/) website (a community-based question-and-answer service). Such applications have attracted the attention of many enterprises, organizations, and individuals seeking the power of GIS. These applications can be used to realize the operation and management of data, to help better enterprises improve efficiency. These demands indicate that GIS development applications have been increasingly sought after, but there are often doubts for those who have not been exposed to GIS applications. Because it is difficult to find the corresponding GIS functions from a large number of functions, some GIS-related assumptions are often not met because a related development technology cannot be found, causing the users to make wrong decisions.

To develop GIS-related applications or tools, developers need to solve many technical problems. They tend to post the questions they encounter on the internet and look forward to answers. Community-based question-and-answer services on the web are attracting more and more scholars and developers to join (Jeon, Croft, & Lee, 2005), such as Stack Exchange (Stack Overflow), Quora, and Reddit. The results of our empirical study show that more than 52.93% of accepted answers, 97.85% of the QGIS-related answers in the whole data sets contain API documentation links. In other words, users often refer to software documentation when answering programming questions (Li, Sun, & Xing, 2018). In this article, we aim to build an information retrieval (IR) system that recommends software documentation for a given programming question.

We can treat the selection of relevant API documentation processes as a ranking task by using a ranking model $f(q, d)$ to sort the documents, where q denotes a query or question and d denotes a document or API documentation in GIS. Traditionally, no training is needed, or only tuning a small number of parameters is necessary (Li, 2011a). Some text-based similarity metrics perform poorly at measuring sentence distance when there is little word overlap between the query and relevant documents, such as the Jaccard coefficient and the overlap coefficient (Nassif, Mohtarami, & Glass, 2016). We need to find an approach that can represent the semantics of document texts.

Compared with open-domain query–API document (Question and Answering (QA)) problems, GIS QA is more challenging. First, the length of each GIS-related API document is usually very long (more than 1,000 words), and the existing open-domain methods are designed for data sets with short text length—such as SQuAD 1.1 (Rajpurkar, Zhang, Lopyrev, & Liang, 2016) and HotpotQA (Yang et al., 2018). When these methods are applied to QGIS data sets, usability is greatly reduced. We tried to use the approach in He, Ning, and Roth (2019) and an approach based on BERT (Devlin, Chang, Lee, & Toutanova, 2018), but failed due to memory overflow issues (we used a machine with 128 GB of memory). Second, the GIS data sets contain a lot of special vocabulary, such as "QgsMapLayer," "QgsLayout," and "QgsGeometry." These words are not frequently used, but they are critical for answering GIS questions. However, the open-domain method cannot represent their semantics well. For example, if we use BERT directly, the word "QgsMapLayer" will be decomposed into "q", "##gs", "##ma", "##play", "##er", losing its special meaning.

To meet the above challenges, we propose an innovative approach called *QA4GIS*. We first use Apache Lucene (https://lucene.apache.org/) and word representations to build a subset from the whole data sets as the candidate documents for each query. Second, we build a four-layer deep learning model to extract the final abstract representation from a query–documentation pair. Third, with the representations of the query–documentation pairs, we train the learning-to-rank (LTR) model using LambdaMART (Wu, Burges, Svore, & Gao, 2010) and use the model to rerank the candidate documents to get the final relevant documents. The key steps are how to extract representations of the query-documentation pairs and training the LTR model. We use a pre-trained DistilBERT (Sanh, Debut, Chaumond, & Wolf, 2019) model to build the sentence representations, Keras (https://keras.io/) to build the four-layer deep learning model, and RankLib (https://sourceforge.net/p/lemur/wiki/RankLib/) to train the LTR model. The innovations of this article are as follows.

1. It is difficult to directly give an answer to a technical question about GIS. Instead, this article turns the QA problem into an IR problem by answering with documents. We created an innovative system with DNN and LTR to answer developer questions on community-based question answering (CQA) websites.
2. To the best of our knowledge, we are the first to use API documents to answer GIS software questions. We also did an empirical study showing that relevant API documentation can answer most QGIS questions.
3. We propose a novel machine learning approach to learn QA pair semantic representations.
4. We did an extensive comparative evaluation and in-depth analysis of QA4GIS.

The remainder of this article is organized as follows. Section 2 summarizes related work. Section 3 presents the empirical study. Section 4 details QA4GIS step by step. Section 5 presents the experiments and evaluation of QA4GIS. Section 6 discusses different aspects of QA4GIS. Section 7 concludes.

2 | RELATED WORK

In this section, we first briefly review some related work on question answering that can be used in the GIS domain. Then we introduce some techniques about learning to rank.

2.1 | Information retrieval

Given a query, an IR system can retrieve documents that are relevant to the query. Queries can represent information needs, and the retrieved documents usually are given a relevance score to the query. Thus, the results of an IR system are typically ranked; this is the key difference of information retrieval searching compared to database searching (Jansen & Rieh, 2010). IR has made a tremendous amount of progress; methods have shifted from traditional bag-of-word retrieval functions such as tf-idf (Wu, Luk, Wong, & Kwok, 2008) and BM25 (Robertson & Zaragoza, 2009), to neural IR models (Cohen, Mitra, Hofmann, & Croft, 2018; Nogueira & Cho, 2017; Rosset, Jose, Ghosh, Mitra, & Tiwary, 2018). Recently, some methods based on BERT have also made significant improvements, for example, MacAvaney, Yates, Cohan, and Goharian (2019) achieves state-of-the-art performance on benchmark data sets using a BERT-based method. In addition, supervised neural ranking models can benefit from weakly labeled data where labels are obtained automatically without human annotators. For example, by labeling the data with BM25 first, Dehghani, Zamani, Severyn, Kamps, and Croft (2017) achieve impressive performance on their neural networks.

2.2 | Question answering

2.2.1 | GIS analytical question answering and geographic question answering

In GIS, a typical workflow of answering an analytical question is to transform a certain type of data into another certain type of goal (Scheider, Meerlo, Kasalica, & Lamprecht, 2020; Scheider, Nyamsuren, Kruiger, & Xu, 2020), which means this kind of question cannot be answered directly. The key to solving these questions becomes finding which types would need to be transformed. Scheider, Ballatore, and Lemmens (2019) turn geospatial questions into a structured query called SPARQL. Scheider, Meerlo, et al. (2020) propose a Web Ontology Language design pattern that can be used for annotating and reasoning over GIS tools. They also formalize how core concepts (including field, object, event, and network) can be represented with common geodata types and how they can be transformed to answer geoanalytical questions.

Geographic questions look very similar to natural language, for example "How long does it take to drive from Newark to New York?" However, they also have their unique differences. First, many geographic questions are highly context-dependent and subjective. Second, the answers are typically derived from a sequence of spatial operations. Third, geographic questions are often affected by vagueness and uncertainty at the conceptual level (Bennett, Mallenby, & Third, 2008; Mai, Yan, Janowicz, & Zhu, 2019). To handle such cases, Mai et al. (2019) use so-called query relaxation and rewriting techniques (Elbassuoni, Ramanath, & Weikum, 2011). Hamzei et al. (2019) propose a data analysis approach that extracts patterns of place-related information through semantic encoding.

In addition, many scholars are very interested in using (knowledge) graphs to deal with geographic questions. Mai et al. (2020) propose a location-aware knowledge graph embedding model called SE-KGE to answer geographic questions in the GIS domain. This model applies a location encoder to incorporate spatial information (coordinates and spatial extent) about geographic entities, and incorporates spatial information (especially spatial extent) about geographic entities into the model architecture. Li, Song, and Tian (2019) use a spatial operation ontology (a formally defined and machine-understandable knowledge base) to categorize data sets and detect the

key elements (space, time, theme, analysis) from a spatial question, then generate an executable workflow for answering the spatial question. Hamilton, Bajaj, Zitnik, Jurafsky, and Leskovec (2018) proposed an end-to-end logic query answering model called GQE which can answer conjunctive graph queries.

2.2.2 | Natural language question answering

Other methods are commonly used in many domains. Brokos, Malakasiotis, and Androutsopoulos (2016) represent documents and questions as weighted centroids of word embeddings and rerank the retrieved documents with word mover's distance (WMD). The WMD measures the dissimilarity between two text documents as the minimum distance that the embedded words of one document need to "travel" to reach another document's embedded words. This method is hyperparameter-free and straightforward to understand and use. We use this method as one of our baselines. Besides, neural ranking networks in the natural language processing domain perform well, and many practitioners apply these models to information retrieval tasks (Mitra, Diaz, & Craswell, 2017). In the work of Xiong, Dai, Callan, Liu, and Power (2017), these methods fall into two groups (Guo, Fan, Ai, & Croft, 2016): *representation-based* and *interaction-based*. Representation-based models aim to obtain a representation for the text in both queries and documents; interaction-based models, on the other hand, aim to capture the textual matching pattern between input texts (Sarvi, Voskarides, Mooiman, Schelter, & de Rijke, 2020). Below are some examples of these two groups.

Representation-based models

Severyn and Moschitti (2016) use a deep learning model for reranking question–answer pairs. Additionally, they combine word embeddings with additional dimensions to encode overlapping words. Guo et al. (2016) view the matching between queries and documents as a nonlinear word transportation problem based on bag-of-word embeddings. DPR (Karpukhin et al., 2020) uses two independent BERT networks (Devlin et al., 2018) as encoders to translate queries and documents into embeddings and uses FAISS (Johnson, Douze, & Jégou, 2017) to retrieve the top k documents.

Interaction-based models

K-NRM (Xiong et al., 2017) uses a translation matrix that models word-level similarities via word embeddings, then uses kernels to extract features and an LTR layer that combines those features into the final ranking score. However, the translation matrix will be huge and the calculation costs will be large when the query or document's length is very long. We also use K-NRM as one of our baselines. Conv-KNRM (Dai, Xiong, Callan, & Liu, 2018) first generates n-gram embeddings using a convolution layer, then forms the translation matrix that can be treated as the ranking signals. However, it is also limited to the length of the query or document.

Most neural models depend on learning useful query and document text embeddings, then use simple similarity metrics like cosine similarity Mitra and Craswell (2018) to find similar documents. However, these models do not work very well on IR tasks due to the massive data sets (Dehghani et al., 2017) and unsuitable word-based representations (Mikolov, Sutskever, Chen, Corrado, & Dean, 2013; Pennington, Socher, & Manning, 2014) for complex, informal search queries. Hence, it is hard to build a model that finds a highly relevant document from a query.

2.3 | Learning to rank

Learning to rank has emerged in the intersection of machine learning, information retrieval, and natural language processing. It is a typically supervised, semi-supervised, or reinforcement learning method that can be used in IR. Learning to rank aims to solve two types of problems: ranking creation and ranking aggregation. Ranking creation

creates a ranking list of documents using the documents' features, while ranking aggregation creates a ranking list of documents using multiple ranking lists (Li, 2011b).

The core steps are encoding query–document pairs into discriminative representations or feature vectors, then using them as the input of an LTR model. The output will be a ranked list of documents with relevant scores.

Depending on the evaluation functions (loss functions) of LTR methods, we can divide the existing LTR methods into the following three categories: pointwise, pairwise, and listwise, as shown in Table 1. *Pointwise* approaches look at a single document at a time in the loss function. They essentially take a single document and train a classifier or regressor on it to predict how relevant it is for the current query (question). *Pairwise* approaches look at a pair of documents at a time in the loss function. Given a pair of documents, they try and come up with the optimal ordering for that pair and compare it to the ground truth. *Listwise* approaches directly look at the entire list of documents and try to come up with the optimal ordering for it.

3 | AN INITIAL EXPLORATORY EMPIRICAL STUDY

3.1 | Motivation

With the growing need to explore spatial data and concepts, some GIS software has become very powerful, with hundreds of functions in the toolbox—for example, ArcMap (Gao & Goodchild, 2013). At the same time, official documents are becoming more and more abundant, making it hard for users to find the function they require. This leads users to have more questions about using GIS software. For example, one question is "Merging multiple vector layers to one layer using QGIS?" The user who asked this question is looking to merge all the layers into one using QGIS. For experienced uses, merging layers using QGIS is often not a problem. However, for novices, it takes much time to find relevant API documents. Therefore, novices tend to seek help from those with experience. If we can help users quickly find the information they need, users can focus on more meaningful domain problems.

3.2 | Research questions and analysis procedure

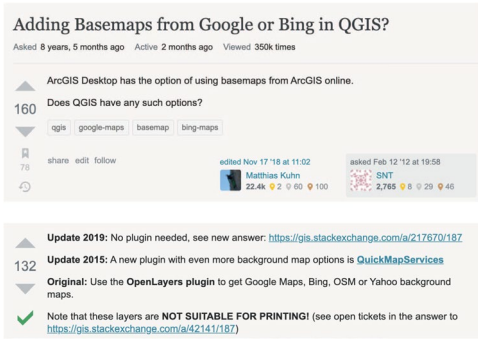
To start the empirical study, we focus on the following research questions. First, how many answers are related to GIS software or tools? Second, how do users answer GIS questions? Third, how can we help users get the information they need faster?

In order to answer these questions, we collected data on the GIS section of the Stack Exchange platform from July 7, 2010, to March 2, 2020, including 121,250 questions with 140,311 answers.

Note that a question may have multiple answers. The user who asked the question has the option to "accept" an answer. Acceptance is indicated by a green checkmark next to the answer. Accepting an answer is not meant

TABLE 1 Three different types of learning-to-rank methods

Type	Approaches
Pointwise	Subset Ranking (Cossock & Zhang, 2006), McRank (Chen, Liu, Lan, Ma, & Li, 2009), Prank (Crammer & Singer, 2002), and OC SVM (Shashua & Levin, 2003)
Pairwise	Ranking SVM (Herbrich, Graepel, & Obermayer, 2000), RankBoost (Freund, Iyer, Schapire, & Singer, 2003), RankNet (Burges et al., 2005), GBRank (Zheng et al., 2008), IR SVM (Cao et al., 2006), Lambda Rank (Burges, Ragno, & Le, 2007), and LambdaMART (Wu et al., 2010)
Listwise	ListNet (Cao, Qin, Liu, Tsai, & Li, 2007), ListMLE (Xia, Liu, Wang, Zhang, & Li, 2008), AdaRank (Xu & Li, 2007), SVM MAP (Yue, Finley, Radlinski, & Joachims, 2007), and Soft Rank (Taylor, Guiver, Robertson, & Minka, 2008)



Question: *Adding Basemaps from Google or Bing in QGIS?*

API documents with links:
- *QuickMapServices* (https://plugins.qgis.org/plugins/quick_map_services/),
- *OpenLayers Plugin* (https://plugins.qgis.org/plugins/openlayers_plugin/).

FIGURE 1 An example of extracting a question–API documentation pair from a discussion thread in the GIS section of Stack Exchange. Note that the answer did not provide a link for *OpenLayers Plugin*; we added it manually

to be a definitive and final statement indicating that the question has now been answered perfectly. It simply means that the author received an answer that worked for them personally. We found that the most frequently occurring tags of questions are QGIS, ArcGIS Desktop, Arcpy, etc. We chose QGIS as our research target because QGIS is the most popular software in this data set. We filtered out all the questions and answers about QGIS and extracted the links inside them. An example is shown in Figure 1. Having manually checked these links one by one, we found the four most frequent kinds of API documents about QGIS from different sources: QGIS API (https://qgis.org/api/), PyQGIS (https://qgis.org/pyqgis/), QGIS Plugins (https://plugins.qgis.org/plugins/), and Documentation for QGIS (https://docs.qgis.org/3.10/en/docs/).

To make the data set with less messy data, we focus on the data according to the following rules (Li et al., 2018): (a) the question has an answer which is accepted as the best answer; and (b) the best answer must contain at least one link to QGIS documentation.

The link in an answer may be ambiguous. For example, one answer suggested the questioner use one QGIS plugin called Qdraw, but the link provided links to the plugin list page. So we manually checked each question–answer pair and API documentation mentioned in the answer. Of the 1,459 unique links, we found that 648 (44.42%) links were marked with a "#" (indicating that these links point to the specific paragraph of the documents). For these documents, we only collected specific paragraph texts. If the link was not specific enough (some answers might want to refer to a paragraph of a web page, but the link does not point to that paragraph), we added an anchor point (which can jump to the specific paragraph) after the link.

3.3 | Results

From Table 2 we can draw the following conclusions. First, 77.97% of all the questions have been answered, which shows that CQA is a very active way for researchers to discuss domain questions. Second, only 32.96% of all the questions have accepted answers, indicating that it is difficult to meet the questioner's expectations. Third, 52.93% of the accepted answers containing links, indicating that people prefer to use links to answer questions.

Figure 2 shows the distributions of numbers of answers to a question and numbers of links in an answer. Figure 2a provides an overview for the whole data set. Due to limited human resources, we cannot crawl through all the GIS software documentation, so we focus on QGIS (which is the most popular tag on the GIS section of Stack Exchange). Figure 2b shows that almost all (97.85%) of the answers have links. In other words, these answers are available in the official QGIS API documents.

TABLE 2 Statistics of the data set

Type	Count	Percentage
All questions	121,250	–
All answers	140,311	–
Questions with answers	94,536	77.97% of all questions
Questions with accepted answers	39,964	32.96% of all questions
Accepted answers	48,029	34.23% of all answers
Accepted answers with links	25,422	52.93% of all accepted answers

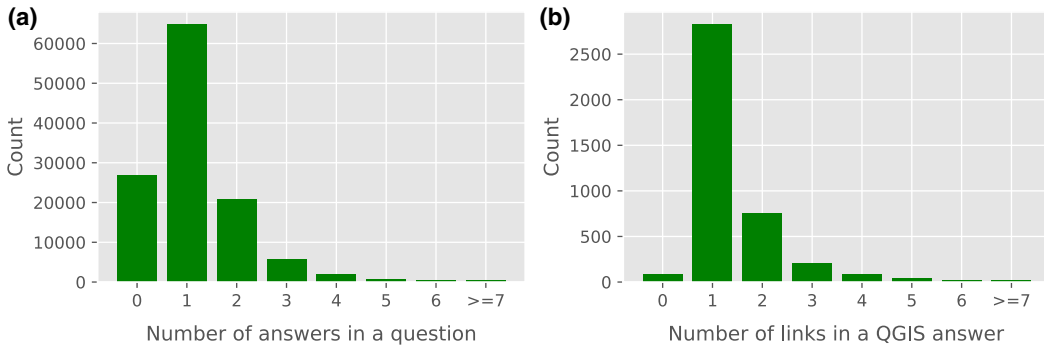


FIGURE 2 Distributions of the number of answers in a question and the number of links in an answer. Graph (a) is based on the whole data set, while (b) focuses on QGIS-related questions. All of these links point to the official documentation of QGIS

In general, this section describes our empirical study on our data set. We can draw the following conclusions (Figure 3). First, about 2,706 answers are related to GIS software tools. Second, users tend to attach API document links to answers (97.85% of the QGIS-related answers have links). The latter conclusion provides a strong argument showing that using API documents to answer GIS developer questions is useful and helpful. Based on this, we came up with a deep learning-based QA system called QA4GIS. In the following section, we show how we built this QA system.

4 | QA4GIS: LEARNING TO ANSWER GIS DEVELOPER QUESTIONS WITH API DOCUMENTATION

In this section we first give an overview of QA4GIS, in which the inputs are questions from developers in natural language, and the outputs are relevant API documentation. We then detail the core steps in QA4GIS to build a QA system.

4.1 | Overview of QA4GIS

As shown in Figure 4, the test phase can be treated as the workflow of our QA system. The training phase indicates how we build each component of QA4GIS step by step. QA4GIS includes four primary steps as follows:

Step 1. *Building question and document representations*. Given a question or document, a language model can translate it into embeddings as the representation. Here we use the pre-trained DistilBERT (Sanh et al., 2019)

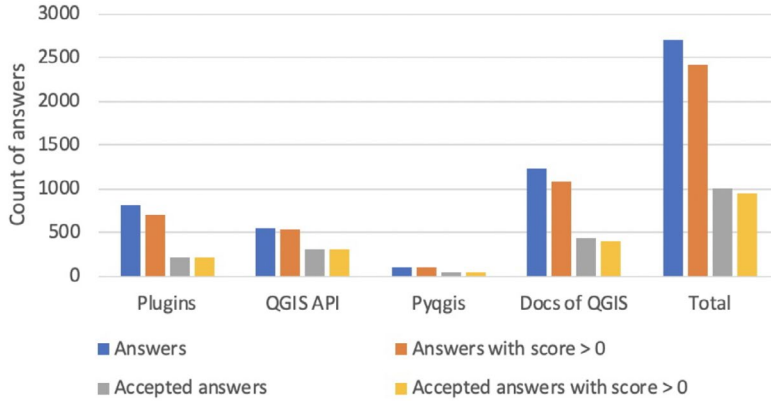


FIGURE 3 Statistics of answers in four types of QGIS documentation

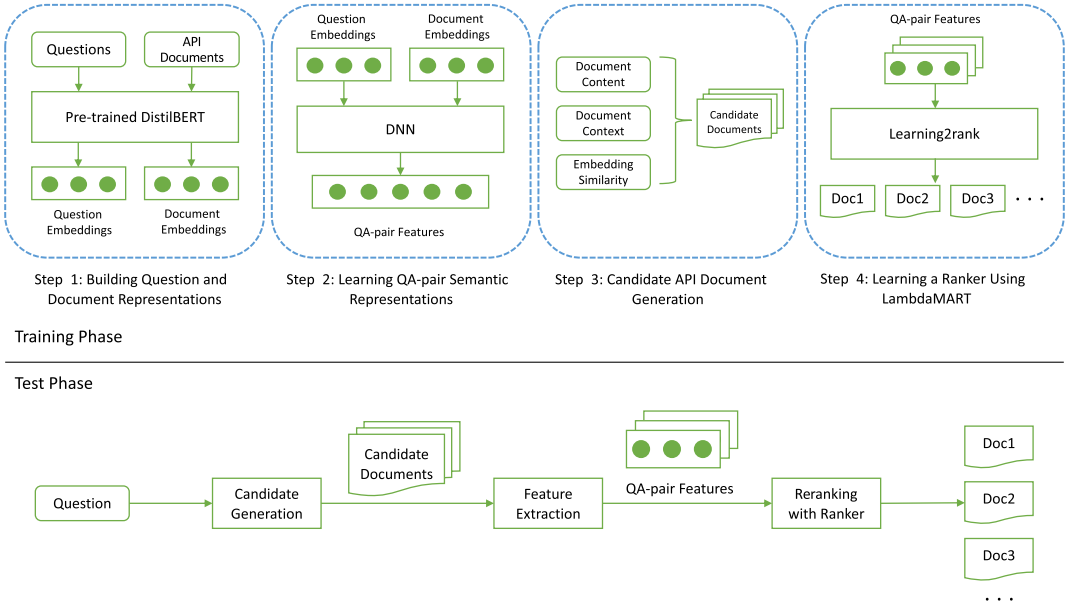


FIGURE 4 The overall architecture of our approach. The training phase aims to translate each question–candidate document pair into abstract representations and learn a ranker with the representations. The testing phase aims to answer the relevant API documents in an order by a given question

model to build question and document embeddings. Note that some common words ("the", "a", "in", "is") that might not add much value to the meaning of the sentence, called "stop-words," need to be removed.

Step 2. *Learning QA pair semantic representations*. Using the question and document embeddings in step 1 as the input, we build a DNN model to learn the QA pair semantic representations.

Step 3. *Candidate API document generation*. Instead of selecting the best relevant API documents from the whole data set, we use the three following methods to build a subset of the entire data set by a given question: API document content, API document context, and embedding similarity. These will be detailed in Section 4.4.

Step 4. *Learning a ranker using LambdaMART*. With steps 2 and 3, we can get the representations of each question–candidate document pair. Then we learn a ranker using LambdaMART to rerank the candidates to get the final relevant API documents in order.

The critical steps are steps 2 and 4. Each step is detailed in the following subsections.

4.2 | Step 1: Building question and document representations

Word embeddings have been widely used to represent natural languages. Compared with traditional methods like one-hot representation, neural language models can capture contextual information. Recently, substantial work has shown that pre-trained models on the large corpus can learn universal language representations, which can avoid the need to train a new model from scratch (Qiu, 2020). In the work of Qiu et al. (2020), many models have been analyzed. Here we choose DistilBERT (Sanh et al., 2019), ELMo (Peters et al., 2018), and GPT-2 (Radford et al., 2019) as our candidate pre-trained language models, and finally select DistilBERT which has the best performance in our experiments.

Although DistilBERT exhibits remarkable performance (Sanh et al., 2019), it still has a shortcoming, which is the maximum sequence length that can be processed (defaults to 512). Thus, for questions and answers that contain a token length of more than 512, we first divide them into several sentences, then use DistilBERT to convert them into embeddings. Finally, we concatenate the embeddints of each sentence to get the embeddings of the entire question or API document. These embeddings of questions and documents can be used as the input to our DNN model in step 2.

4.3 | Step 2: Learning QA pair semantic representations

In this step we build a DNN model to learn the semantic representation for QA pairs. As shown in Figure 5, it has sentence embeddings, a convolution layer, a pooling layer, a join layer, hidden layers, and a softmax layer.

The DNN model's input are the sentence embeddings of a question and an API document. We capture the embeddings of each sentence. For example, if there is a document that has n sentences, and each sentence is represented as a m-dimensional vector, then the sentence embeddings of this document will be an $n \times m$ matrix.

To better capture GIS-related information and patterns, we weight the sentence embeddings (see Figure 5). Specifically, we extracted 10,869 technical terms related to GIS and QGIS including API names, plugin names. and common attributes, from our data set. If a related term appears in a sentence, then this sentence will be given a weight w, and the embeddings of the sentence will be multiplied by w. The weight w will be treated as a DNN parameter during the training.

We calculate the maximum length of the question and the API document separately. For those questions or API documents with fewer words than the maximum length, we use zero-padding to fill them in. Zero-padding has only a minor effect and has been tested in Dwarampudi and Subba Reddy (2019). In addition, in the field of NLP, there are many other papers using zero-padding, e.g., Cliché (2017). The DNN model's output consists of two values: the probabilities that the query and document are related and unrelated. Here we say if the query and document are related, the query can be answered by this document. We have a two-dimensional output here because the softmax layer must have classifications (Bridle, 1990).

Note that the procedures to get the query and document representation are the same before the join layer.

4.3.1 | Convolution layer

The convolution layer aims to extract patterns from query embeddings or document embeddings.

Given the input sentence matrix $S \in \mathbb{R}^{d \times |s|}$ (where $|s|$ is the count of sentences in a question or document) and a filter $F \in \mathbb{R}^{d \times m}$, the output will be a vector $v \in \mathbb{R}^{|s|-m+1}$. Each component of v is computed as:

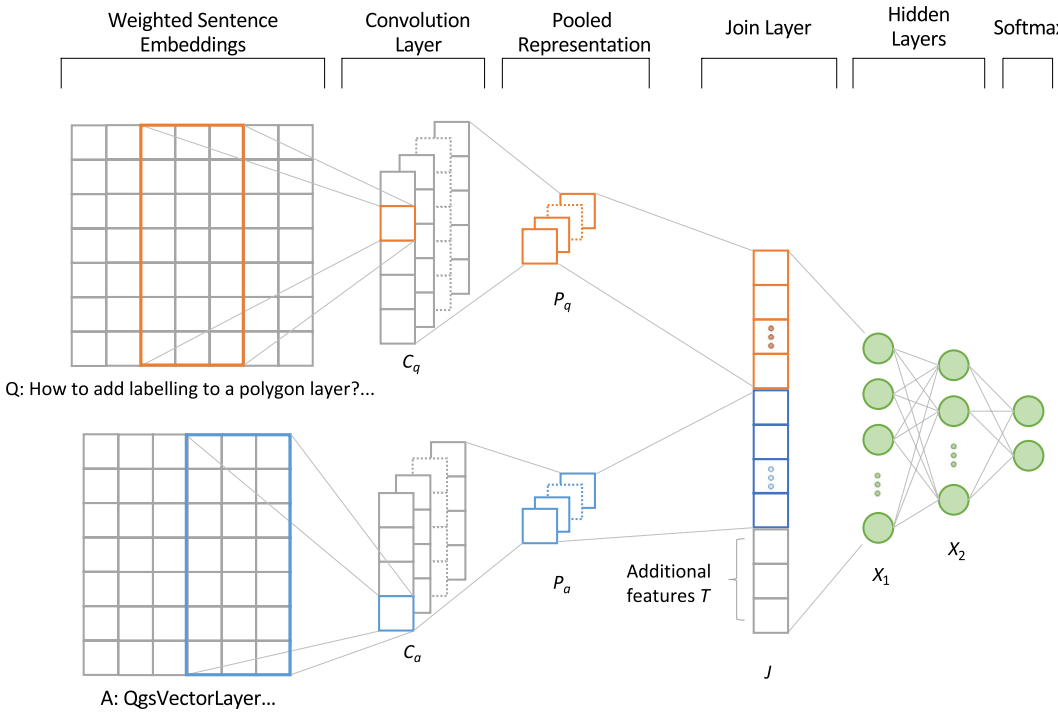


FIGURE 5 Deep neural network for QA pair semantic representation

$$v_i = (S * F)_i = \sum_{k,j} (S_{[:,i:i+m-1]} \otimes F)_{k,j} \tag{1}$$

where * is the convolution operation, is elementwise multiplication, $S_{[:,i:i+m-1]}$ $(m \le i \le |s| - m + 1)$ is a matrix slice of size m along the columns, and k, j are iterators of $S_{[:,i:i+m-1]}$ and F on rows and columns, respectively. Note that the convolution filter F is of the same dimensionality d as the input sentence matrix S, both equal to the dimensionality of each word embedding in Section 4.2.

To form a richer representation of the sentence, we apply a set of filters that work in parallel, generating multiple vectors v—feature maps V. In practice, we also need to add a bias vector $b \in \mathbb{R}^n$ (where n is the number of filters) to V; the result is temporarily named E. This allows the model to learn an appropriate threshold. To learn nonlinear decision boundaries, we use *ReLU* as the activation function. Then, each component of the final output C of the convolution layer can be written, for $i \in \{1, \ldots, n\}$ and $j \in \{1, \ldots, |s| - m + 1\}$, as:

$$\begin{aligned} E_{i,j} &= V_{i,j} + b_i \\ C_i &= \mathrm{ReLU}(E_i) = \max(0, E_i) \end{aligned} \tag{2}$$

4.3.2 | Pooling layer

A pooling layer is used to progressively reduce the representation's spatial size to reduce the number of parameters and computation in the DNN model. We use max-pooling in our model. This operates on the columns of the feature map matrix C, returning the maximum value $pool(C_i) : \mathbb{R}^{|s|-m+1} \to \mathbb{R}$, where $i \in \{1, \ldots, n\}$. The output vector P of the pooling layer can be written as:

$$P = \begin{bmatrix} \max(C_1) \\ \dots \\ \max(C_n) \end{bmatrix} \tag{3}$$

Below, we use P_q and P_a to denote the representations of query and API document, respectively.

4.3.3 | Join layer

In the join layer we combine the representations of query and document. Also we add some straightforward content features T, such as word overlap count, word overlap count weighted by inverse document frequency (IDF), and BM25 score (Robertson & Zaragoza, 2009) (detailed in Section 5.1) between query and document. Then the output J of the join layer can be written as.

$$J = [P_q, P_a, T]. \tag{4}$$

While neural models have a large capacity to learn complex decision functions, they tend to overfit, especially on small and medium-sized data sets. To mitigate the overfitting issue, we augment the cost function with L2-norm regularization terms for the model's parameters (Severyn & Moschitti, 2015).

4.3.4 | Hidden layers

The hidden layers are two fully connected layers. They operate on the results of the join layer. We use two layers instead of one because two layers can achieve a better abstraction in our experiments. Each output of these two hidden layers X_1, X_2 with parameters W_1, b_1, W_2, b_2 can be represented as:

$$\begin{aligned} X_1 &= W_1 \cdot J + b_1 \\ X_2 &= ELU(W_2 \cdot X_1 + b_2) \end{aligned} \tag{5}$$

where *ELU* is the activation function of the second hidden layer, calculated as:

$$ELU(x) = \begin{cases} x, & \text{if } x > 0 \\ \text{alpha}^*(\exp(x) - 1), & \text{if } x < 0 \end{cases} \tag{6}$$

where alpha is a coefficient set to 1.0 by default. The second hidden layer $X_2 = \{x_1, x_2, \dots, x_m\}$ can be used as the final abstract representation of the question–document pair. We will use it to learn a ranker. The output layer uses softmax as an activation method. It has two output values, which represent the probabilities that the question and the document are related and unrelated.

4.4 | Step 3: Candidate API document generation

Because there is a great deal of API documentation, it is not easy to directly select the best relevant API document from the whole data set. Instead, we use the three methods below to build a subset of the whole data set—candidate API documents.

4.4.1 | API document content

We build a search engine for software documentation using Apache Lucene. Lucene can index all the API documents. Given a query, it will return the documents containing keywords. Specifically, stop-word removal and stemming are performed as pre-processing. For each query, the search engine returns the 10 most relevant results.

4.4.2 | API document context

The discussions on Stack Exchange provide enriching context to mine usage scenarios of API documents. When an API document appears in a discussion thread, its surrounding texts reflect its relevance to the question. Since an API document can be mentioned in multiple threads, we collected all the surrounding texts from different answers about each document. That is, given a query, we search the answers containing keywords and returning the API documents mentioned in the answers. For each query, we also use Apache Lucene to get the 10 most relevant results.

4.4.3 | Embedding similarity

We use the language model trained in step 1 to translate each word in questions and documents into embeddings. By averaging word embeddings in a sentence, we can get the sentence embeddings. Then, given a query, we retrieve the top 10 API documents based on the cosine similarity between the question embeddings and API document embeddings. Cosine similarity is calculated as:

$$sim(x, y) = \frac{x \cdot y}{\|x\| \, \|y\|} \tag{7}$$

where $\|x\|$ is the Euclidean norm of vector $x = (x_1, x_2, \ldots, x_n)$, defined as $\sqrt{x_1^2 + x_2^2 + \ldots + x_n^2}$ (and similarly for $\|y\|$). The closer the cosine similarity value to 1, the greater the match between vectors.

Then we merge the related API documents searched by the above methods. Since there may be an intersection among them, we will get up to 30 results. For example, given a query "Choropleth of calculated Voronoi polygons?" we may get the following related API documents: {qgis_docs_print_composer, qgis_docs_geometry_tools, qgis_docs_geometry,...}. Note that the order in which they appear can be ignored.

4.5 | Step 4: Learning a ranker using LambdaMART

The last step of our approach is to learn a ranker to rerank the candidate API documents. As described in Section 4.3.4, the output of the second hidden layer of the DNN can be thought of as the final abstract representation X of a query–API document pair. Our goal is to build a ranking model $f: X \rightarrow \mathbb{R}$ trained to map the vector X to a real-valued score such that, for a given query, more relevant documents are scored higher among all the candidate documents.

To build such a model, we briefly study some other existing LTR approaches. As shown in Table 1, LambdaMART (Wu et al., 2010) achieved the best performance in the recent Yahoo! Learning to Rank Challenge. So we use LambdaMART to learn such a ranker. As shown in Figure 6, the training data (input) of the LTR model consists of n instances; each instance contains one query q, the representations $X = (x_1, x_2, \ldots, x_m)$ of each query–document pair, and a label y (1 for related, 0 for unrelated). The output of the LTR model are ranked API documents with relevance scores. The higher the score, the higher the ranking.

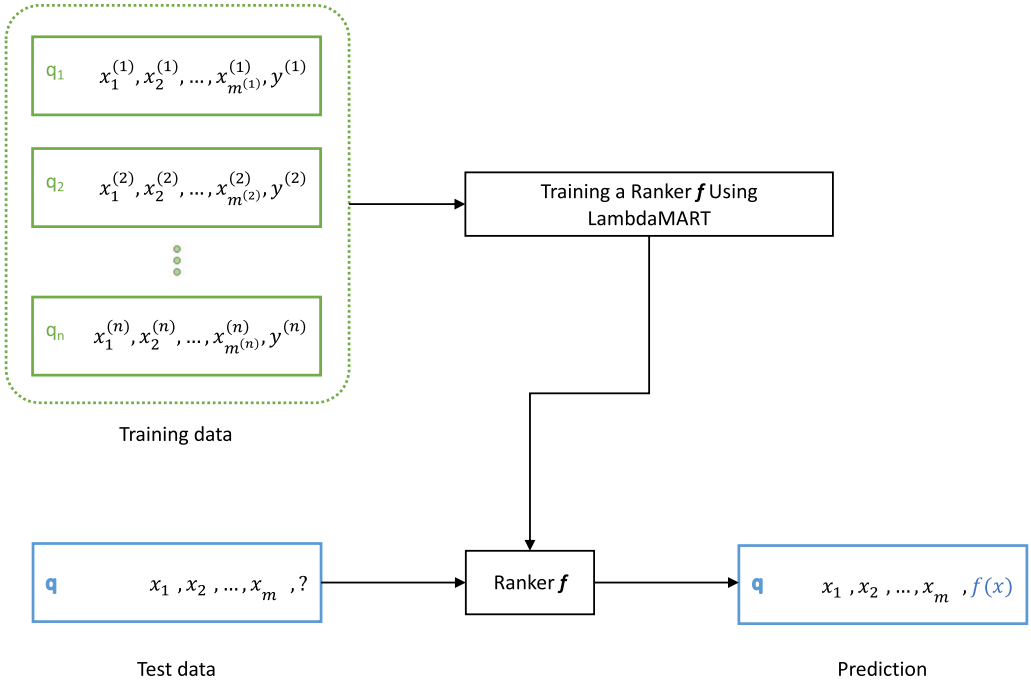


FIGURE 6 Learning a ranker to re-rank the candidate API documents

5 | EMPIRICAL EVALUATION

5.1 | Research questions

To evaluate QA4GIS, we seek to answer the following research questions:

RQ1. *Overall QA comparability study*. How does QA4GIS compare to other state-of-the-art methods in performance?

RQ2. *Sensitivity analysis of QA4GIS*. How do various factors affect the overall performance of QA4GIS?

RQ3. *Qualitative analysis of QA4GIS*. How well does QA4GIS perform in answering real questions?

5.2 | Data set

We collected data on the Stack Exchange platform's GIS section from July 7, 2010 to March 2, 2020, including 121,250 questions with 140,311 answers. For the API documents mentioned in the questions and answers, there are four most frequent kinds of API documents about QGIS from different sources: QGIS API (https://qgis.org/api/), PyQGIS (https://qgis.org/pyqgis/), QGIS Plugins (https://plugins.qgis.org/plugins/), and Documentation for QGIS (https://docs.qgis.org/3.10/en/docs/). We wrote a spider using Python to crawl all of these.

Since the content and structure of these four types of documents are different, we process them separately. QGIS API mainly introduces various libraries of QGIS, most of which are descriptions of namespaces, classes, and functions in the code. We need to remove a large number of meaningless variable declarations, such as “double u” and “int I”. PyQGIS is similar to QGIS API, but has a different content structure. Here again we need to remove the variable declarations. QGIS Plugins contains 1,266 plugin introductions, and some plugins provide other web pages to introduce the uses. So we have to organize the introduction of each plugin manually. Documentation for

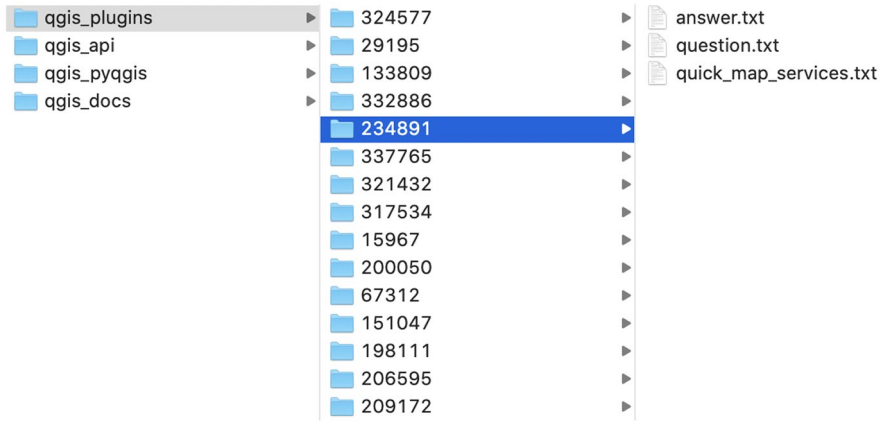


FIGURE 7 The structure of our data set. Folder numbers are question IDs

QGIS is the most detailed documentation in QGIS, with rich text and graphic tutorials. The link to it often contains "#", so we need to extract the corresponding paragraphs.

As shown in Figure 7, we organize our data as follows. Four folders correspond to API documents from the four sources. In each folder are lots of subfolders named with question-IDs that include the current type of documents. Each question folder contains a question, answer, and API documents mentioned in the answer (named with the title of API document). Note that there may be more than one document mentioned in an answer.

5.3 | Evaluation methodology

5.3.1 | RQ1: Overall QA comparability study

Model training and settings

This subsection details how we train our models in QA4GIS, build the input data and specify the hyperparameters. There are two models that need to be trained: the DNN model and the LTR model.

The training data for the *DNN* model consists of positive samples (labeled 1) and negative samples (labeled 0). The positive samples are the ground-truths (e.g., if an API document is mentioned in the accepted answer of a question, then it is one of the ground-truths), while the negative samples are the documents that are unrelated to the question at hand. We do this to learn a loss function that reduces the distance between a question and its positive samples, but increases the distance between a question and its negative samples. Given a question, we consider two ways to generate negative samples: randomly selecting the documents which appear in the training set; using BM25 (Robertson & Walker, 1994; Robertson & Zaragoza, 2009) to retrieve documents which do not contain the answer but match question tokens heavily.

BM25 is a probabilistic model representing the relevance of document d to query q. It is calculated as:

$$BM25(q,d) = \sum_{w \in q \cap d} idf\ (w)\ \frac{(k+1)\,tf\ (w)}{tf\ (w)\ + k\left((1-b)\ + b\,\frac{dl}{avgdl}\right)} \tag{8}$$

where w denotes a word in d and q, $tf\ (w)$ denotes the frequency of w in d, $idf\ (w)$ denotes the inverse document frequency of w, dl denotes the length of d, $avgdl$ denotes the average document length, and k and b are parameters. We take $k = 1.5, b = 0.75$ in our experiments. We chose BM25 because it is an excellent traditional method and does not require training.

Let $D = \{ \langle q_i, d_{i,m}^+, \ldots, d_{i,1}^+, d_{i,1}^-, \ldots, d_{i,n}^- \rangle \}_{i=1}^k$ be the training data, consisting of k instances. Each instance contains one question q_i and m relevant (positive) documents $d_{i,j}^+$, along with n irrelevant (negative) documents $d_{i,j}^-$. For the DNN model, the ratio between positive samples and negative samples is 1: 1. With the trained language model, we can translate these samples into embeddings as the DNN model's input.

Our training parameters are set as follows: the dimension of each sentence embedding is 768, the weight of sentences with GIS-related terms is 1.5, the number of filters and kernel size (specifying the length of the convolution window) in the convolution layer are 64 and 5 respectively, the first hidden layer size is 128, and the second hidden layer size is 64.

For the *LTR* model, we also need positive and negative samples to train it. In this case, we prepare the training data according to three different rates between positive and negative samples: m: 1, m: 10, m: 30, where m is the number of ground truths in an instance. Then using these samples as the input of the learned DNN model, we can extract the second hidden layer's output from the DNN model as the features of the question–API documents. These features are the input of the LTR model. We use Ranklib (https://sourceforge.net/p/lemur/wiki/RankLib/) to implement our LTR model. The detailed data format description can be found here (https://sourceforge.net/p/lemur/wiki/RankLib%20File%20Format/).

Evaluation metrics and baselines

The *metrics* used to evaluate the quality of our ranker model and baselines are precision at k ($P@k$), recall at k ($R@k$), mean average precision (*MAP*), and mean reciprocal rank (*MRR*), which are common in information retrieval and question answering tasks.

- $P@k = \left| D_k \cap D_g \right| / k$, is the fraction of API documents relevant to the query question among the top k ranked results. D_k denotes the set of top k ranked API documents and D_g is the set of ground-truth API documents.
- $R@k = \left| D_k \cap D_g \right| / \left| D_g \right|$ is the fraction of ground-truth API documents in the top k results.
- $MAP(Q) = \frac{1}{|Q|} \sum_{j=1}^{|Q|} AveP(j)$, where $AveP(j)$ is the average prediction (AP) for query j. The AP uses a combination of the precision at successive sub-lists, combined with the change in recall in these sub-lists (https://en.wikipedia.org/wiki/Evaluation_measures_(information_retrieval)). Q is the set of test queries. This metric is able to give more weight to errors that happen high up in the recommended lists. Conversely, it gives less weight to errors that happen deeper in the recommended lists. This matches the need to show as many relevant items as possible high up the recommended list.
- $MRR(Q) = \frac{1}{|Q|} \sum_{j=1}^{|Q|} \frac{1}{rank(j)}$, where $rank(j)$ is the position of the first relevant API document in the candidate list, Q is the set of test queries. MRR only focuses on the rank of the first relevant API document. Hence it is more suitable in cases where there is only one API document related to the question.

To answer the RQ1, we chose the following essentially different methods as our *baselines*.

- *Lucene* (https://lucene.apache.org/), Lucene is an open-source search engine developed by the Apache Software Foundation. Given a query, we use Lucene to search the API documents, and it returns top k ranked results.
- *Okapi BM25* (Robertson & Zaragoza, 2009). BM25 is a traditional method that is a bag-of-words retrieval function that ranks a set of API documents based on the query terms appearing in each document.
- *Word2vec* (Mikolov et al., 2013). We use Word2vec to build the word embeddings and calculate the cosine similarity between each query and API document. The closer the cosine similarity score is to 1, the better the match between the document and the question.
- *Word mover distance (WMD)* (Brokos et al., 2016). Similar to Word2vec, WMD is based on word embeddings, but it focuses on the distance between embeddings. The distance between query and API document is calculated

by the minimum cumulative distance that words from the query need to travel to match the point cloud of the API document.

- *K-NRM* (Xiong et al., 2017). K-NRM is an interaction-based method. Given the query words and API document words, K-NRM calculates the word–word similarities and forms the translation matrix. It then extracts ranking features from the translation matrix and uses the LTR model to combine the features to get the final ranking score.
- *Pre-trained ELMo* (Peters et al., 2018). ELMo is a deep contextualized word representation model. This baseline uses a pre-trained ELMo model (https://tfhub.dev/google/elmo/3) to build embeddings for each question and document. Since the pre-trained ELMo model cannot handle a document which is larger than the default maximum sequence length of the model, we first split each document into multiple sentences, then use ELMo to calculate the embedding of each sentence and take the average value as the embedding of the entire document, and finally calculate the cosine similarity between the questions and API documents.
- *Pre-trained DistilBERT* (Sanh et al., 2019). DistilBERT is a distilled version of BERT (Devlin et al., 2018). This baseline uses a pre-trained DistilBERT model (https://huggingface.co/transformers/model_doc/distilbert.html) to build embeddings for each question and document. We proceed in the same way as pre-trained ELMo to build document embeddings, then calculate the cosine similarity between the questions and API documents.
- *Pre-trained GPT-2* (Radford et al., 2019). GPT-2 is a large transformer-based language model with 1.5 billion parameters. This baseline uses the pre-trained GPT-2 mode (https://huggingface.co/transformers/model_doc/gpt2.html) to build embeddings for each question and document. We proceed in the same way as pre-trained ELMo to build document embeddings, then calculate the cosine similarity between the questions and API documents.
- *SIF* (Arora, Liang, & Ma, 2017). SIF is a method to calculate sentence embeddings. It first uses a pre-trained language model to get word vectors, then modifies the weights by using the random walk model (Arora et al., 2017). We use this baseline to get the embeddings of each question and document, and then calculate the cosine similarities.

5.3.2 | RQ2: Sensitivity analysis of QA4GIS

QA4GIS has four components, namely the language model, DNN model, candidate documents generator, and ranker. The first three components are for the last component, the ranker. For the DNN model, we mainly fine-tune the parameters until we find the best parameters. The candidate documents generator is mainly used to narrow the scope of the API documents. If we directly run the ranker on the whole data set, we need a huge amount of calculation, but the impact of final rank results is minimal.

As the input of the DNN model, the amount of semantic information contained in sentence embeddings will greatly affect the results of the experiment. Therefore, we selected four language models and tested their influence on the experimental results under the same data set. These language models are: Mikolov's skip-gram model (also known as Word2vec) (Collobert et al., 2011; Turian, Ratinov, & Bengio, 2010), DistilBERT (Sanh et al., 2019), ELMo (Peters et al., 2018), and GPT-2 (Radford et al., 2019). Note that to get good training performance, DistilBERT, ELMo, and GPT-2 require a much larger data set than the one we manually collected. Thus we use the pre-trained models (as same as the baseline experiments) instead of training by ourselves.

Learning to rank is an essential component of QA4GIS, which directly affects the final result. We have chosen the following three LTR factors to observe how the final performance will be affected if modified (the two methods to generate negative samples were detailed earlier): (1) the number of negative samples; (2) the method (random or BM25) to generate negative samples; and (3) whether to use an LTR model.

5.3.3 | RQ3: Qualitative analysis of QA4GIS

To more intuitively reflect the performance of QA4GIS, we will compare QA4GIS with the best baseline by some real examples. Specifically, given an API-related question, we use each of them to recommend the top 30 API documents in order, and then compare the ranking of the ground-truth documents in the two sets of results. The higher the ranking, the better the performance.

5.4 | Evaluation results

5.4.1 | Results of RQ1: Overall QA comparability study

The ranking performance of each ranking method on the GIS data set is shown in Table 3. In each evaluation metric, the best score is highlighted in bold. Our QA4GIS approach outperforms all other baselines on all evaluation metrics. Below each QA4GIS metric score, we calculate the percentage improvement of our method over the second-best method. Specifically, compared with the best performance baseline SIF, we improve 21.39% on the MAP score and 22.34% on the MRR score. We even improved 54.65% on the $P@10$ metric.

Observe that as k grows, $P@k$ becomes smaller on all the methods, including our approach. This phenomenon is related to the number of ground-truth documents. On Stack Exchange, 70.46% of the answers contain only one link to the API document (reported in Figure 2b). As a result, $\left|D_k \cap D_g\right| = 1$ in most cases. Thus, the ideal value of $P@k$ is slightly above $1/k$ when $k > 1$. So it is not surprising that $P@10$ is very small for all the methods.

5.4.2 | Results of RQ2: Sensitivity analysis of QA4GIS

Impact of the different language models

As can be seen from the Table 4, different language models have quite different experimental results. The best experimental result is DistilBERT, whose MAP and MRR are 0.3819 and 0.3971, respectively. The worst experimental result is GPT-2, scoring only 0.1492 and 0.1551. Compared with Word2vec, DistilBERT has improved by 30.07% and 35.76% in terms of MAP and MRR, respectively.

Impact of the number of negative samples

Table 5 shows that the number of negative samples can affect the performance of our approach. We tested three different numbers of negative samples: $\{1, 10, 30\}$. As shown in Table 5, adding more negative samples does not improve the MAP and MRR scores. On the contrary, it can lower the scores. Note that in this experiment, the language model we used is Word2vec, the method to generate negative samples is BM25, and the LTR model is LambdaMART.

Impact of the method to generate negative samples

Table 6 shows that the different methods to generate negative samples can also affect the performance of our approach. Compared with random selection, using BM25 to generate negative samples can achieve improvements of 42.87% on the MAP score and 43.25% on the MRR score. Because it uses documents that are close to but not equal to the ground-truths, the LTR model can better learn the features which are more conducive to describing the document. Note that in this experiment, the language model we used is Word2vec, the number of negative samples is 1, and the LTR model is LambdaMART.

TABLE 3 Results of RQ1. Performance of *P @ k*, *R @ k*, *MAP* and *MRR* for different methods

Method	P@1	P@3	P@5	P@10	R@1	R@3	R@5	R@10	MAP	MRR
Lucene	0.1026	0.0641	0.0436	0.0269	0.1026	0.1923	0.2179	0.2692	0.1586	0.1586
Word2vec	0.0513	0.047	0.0436	0.0462	0.0513	0.141	0.2179	0.4615	0.1697	0.169
BM25	0.0897	0.047	0.0385	0.0359	0.0897	0.141	0.1923	0.3526	0.1881	0.1872
WMD	0.1154	0.0556	0.041	0.0397	0.1154	0.1667	0.2051	0.3974	0.2094	0.2088
K-NRM	0.1154	0.0598	0.0667	0.0538	0.1154	0.1731	0.3269	0.5256	0.2333	0.2355
DRMM	0.0760	0.0960	0.0776	0.0640	0.0680	0.2505	0.3418	0.5556	0.2284	0.2357
Pre-trained ELMo	0.0670	0.0499	0.0431	0.0394	0.0575	0.1297	0.1830	0.3361	0.1586	0.1683
Pre-trained GPT-2	0.0793	0.0775	0.0722	0.0577	0.0667	0.2049	0.3171	0.4972	0.2101	0.2196
Pre-trained DistilBERT	0.1120	0.0917	0.0774	0.0617	0.1037	0.2513	0.3489	0.5459	0.2466	0.2538
SIF	0.1913	0.1185	0.0935	0.0666	0.1753	0.3245	0.4176	0.5900	0.3146	0.3246
QA4GIS	**0.1997** (↑ 4.39%)	**0.1653** (↑ 39.49%)	**0.1410** (↑ 50.80%)	**0.1030** (↑ 54.65%)	**0.1754** (↑ 0.06%)	**0.4278** (↑ 31.83%)	**0.6056** (↑ 45.02%)	**0.8685** (↑ 47.20%)	**0.3819** (↑ 21.39%)	**0.3971** (↑ 22.34%)

Note: Note that the best performance is highlighted in bold. The final row is the percentage improvement of our method QA4GIS over the method SIF which achieved the second-best performance.

TABLE 4 Results of RQ2. MAP and MRR performance of *QA4GIS* under different language models

Language models	MAP	MRR
Word2vec	0.2936	0.2925
Pre-trained DistilBERT	**0.3819**	**0.3971**
Pre-trained ELMo	0.2849	0.2984
Pre-trained GPT-2	0.1492	0.1551

Note: The best performance is highlighted in bold.

TABLE 5 Results of RQ2. MAP and MRR performance of QA4GIS under different numbers of negative samples

Number of negative samples	MAP	MRR
1	**0.2936**	**0.2925**
10	0.2405	0.2390
30	0.2264	0.2255

Note: The best performance is highlighted in bold.

TABLE 6 Results of RQ2. MAP and MRR performance of QA4GIS under different methods to generate negative samples

Method	MAP	MRR
Random	0.2055	0.2042
BM25	**0.2936** (↑ 42.87%)	**0.2925** (↑ 43.25%)

Note: The best performance is highlighted in bold. The percentage after the best score is the percentage of improvements compared with another method.

Impact of whether to use an LTR model

Table 7 shows that using an LTR model can significantly improve performance. Compared with not using an LTR model, LambdaMART can achieve improvements of 75.39% on the MAP score and 74.42% on the MRR score. Note that in this experiment, the language model we used is Word2vec, the number of negative samples is 1, and the method to generate negative samples is BM25.

We have also experimented with several other factors for our approach, such as using more LTR model features to rank models and fine-tuning the DNN model parameters. However, we only observed similar or worse performances. Thus, we chose to present the simplest successful model to illustrate its effectiveness better.

5.4.3 | Results of RQ3: Qualitative analysis of QA4GIS

We chose three real questions from the Stack Exchange website to compare QA4GIS with SIF. These questions are: "Choropleth of calculated Voronoi polygons?", "GIS dataset with latitude, longitude, and elevation", and "Extracting shapefile attributes from raster colors using PyQGIS?" As shown in Tables 8–10, the ground-truth documents all rank first in the QA4GIS results, while they rank 23, after 30, and 15 respectively in SIF.

The difference between QA4GIS and SIF

It is evident that QA4GIS can achieve a better performance than SIF. Although both methods learn sentence embeddings, the relationship between the questions and the API documents is different under the two methods. What

TABLE 7 Results of RQ2. MAP and MRR performance of QA4GIS regarding whether to use a learning-to-rank model

Learning-to-rank model	MAP	MRR
None	0.1674	0.1677
LambdaMART	**0.2936** (↑ 75.39%)	**0.2925** (↑ 74.42%)

Note: The best performance is highlighted in bold. The percentage after the best score is the percentage of improvements compared with another method.

SIF learns is an independent sentence representation, and there is no relationship between questions and documents. QA4GIS learns how to represent each pair of questions and documents. Any question and any document can be expressed as embeddings of a fixed dimension. SIF has a very good performance (compared to BERT) when calculating the similarity between documents and problems. However, the high similarity between the document and the question does not mean that the document can answer the user's question, because a technical problem often has multiple solutions. Therefore, QA4GIS, which can learn and extract the features of the question–document pair, performs better.

6 | DISCUSSION

In this section we discuss the advantages, limitations and some applications of QA4GIS, we also discuss whether our method can be used for other data sets.

6.1 | The advantages of QA4GIS

First, compared with traditional methods, such as Lucene and BM25, QA4GIS can find the relevant documents even if there is no identical word in the question and the document. Second, compared with word embedding methods, such as Word2vec, ELMo, GPT-2, and BERT, QA4GIS can extract the features from questions and document embeddings using a convolution layer. Using the extracted features instead of the original word embeddings is an improvement we made based on the above word embedding methods. Third, compared with K-NRM, which is very similar to QA4GIS, we use additional features to describe the relationship between the question and document, and use three methods to generate candidate answers to improve the hit rate.

6.2 | The limitations of QA4GIS

First, our approach does not work well when the questions contain too many codes and few natural language words. Because the variable names in the code are often abbreviations of words, they are meaningless. Second, our approach does not work well on questions that are too general. Without a more detailed description, it is difficult for even experts to give appropriate recommendations.

6.3 | Applications of QA4GIS

With QA4GIS, one direct application is that we can build a GIS information retrieval website where users can query the API documents they need in natural languages. The second application is a plugin based on

TABLE 8 Results of RQ3. The API documentations recommended by QA4GIS and SIF for the question "*Choropleth of calculated Voronoi polygons?*"

Rank	QA4GIS	SIF
1	qgis_docs_print_composer ✓	qgis_api_qgszonalstatistics_class_reference
2	qgis_docs_geometry_tools	qgis_docs_information
3	qgis_docs_geometry	qgis_docs_a_gentle_introduction_to_gis
4	qgis_docs_clipping_and_merging_raster_layers	qgis_docs_extract_vertices
5	qgis_docs_modifying_vector_layers	qgis_docs_raster_to_vector_conversion
6	qgis_docs_geometry_functions	qgis_docs_topological_editing
7	qgis_pyqgis_length	qgis_docs_editing
8	qgis_docs_default_values	qgis_docs_zonal_statistics_plugin
9	qgis_docs_field_calculator	qgis_docs_general_tools
10	qgis_docs_geometry_predicates_and_operations	qgis_docs_working_with_the_attribute_table
…	…	…
23	qgis_plugins_AutoFields	qgis_docs_print_composer ✓
…	…	…

Note: The underline and checkmark (✓) after the API document title indicates that the document is ground-truth.

TABLE 9 Results of RQ3. The API documentations recommended by QA4GIS and SIF for the question "*GIS dataset with latitude, longitude, and elevation.*"

Rank	QA4GIS	SIF
1	qgis_docs_gdal2xyz ✓	qgis_docs_query_builder
2	qgis_pyqgis_qgsgpsinformation	qgis_api_qgsrastercalculator_class_reference
3	qgis_api_qgsgpsinformation_struct_reference	qgis_docs_topological_editing
4	qgis_docs_introducing_gis	qgis_docs_raster_calculator
5	qgis_docs_coordinate_reference_systems	qgis_docs_raster_to_vector_conversion
6	qgis_docs_coordinate_reference_system_crs_in_detail	qgis_docs_general_tools
7	qgis_docs_custom_coordinate_reference_system	qgis_docs_information
8	qgis_docs_inverse_distance_weighted_idw	qgis_docs_symbology_properties
9	qgis_docs_changing_raster_symbology	qgis_docs_setting_the_snapping_tolerance_and_search_radius
10	qgis_docs_importing_a_delimited_text_file	qgis_docs_heatmap_plugin
11	qgis_docs_default_values	qgis_docs_using_spatial_index
…	…	…
30	qgis_docs_loading_a_layer_from_a_file	qgis_docs_importing_a_delimited_text_file

Note: The underline and checkmark (✓) after the API document title indicates that the document is ground-truth.

Google Chrome. When a user enters the GIS section of the Stack Exchange website and uses the search function developed by the Stack Exchange, the plugin can search simultaneously and give suggested API documents.

TABLE 10 Results of RQ3. The API documentations recommended by QA4GIS and SIF for the question "*Extracting shapefile attributes from raster colors using PyQGIS?*"

Rank	QA4GIS	SIF
1	qgis_docs_using_raster_layers ✓	qgis_docs_list_of_functions
2	qgis_docs_changing_raster_symbology	qgis_docs_status_bar
3	qgis_docs_using_pyqgis_in_custom_applications	qgis_docs_using_vector_layers
4	qgis_docs_iterating_over_vector_layer	qgis_docs_rule_based_renderer
5	qgis_docs_symbology_properties	qgis_docs_editing_attribute_values
6	qgis_pyqgis_iterating_over_vector_layer	qgis_docs_setting_the_snapping_tolerance_and_search_radius
7	qgis_docs_graduated_renderer	qgis_pyqgis_geometry_handling
8	qgis_docs_zonal_statistics_plugin	qgis_docs_geometry
9	qgis_docs_raster_terrain_analysis_plugin	qgis_docs_field_calculator
10	qgis_docs_pyqgis_developer_cookbook	qgis_docs_working_with_projections
...	...	...
15	qgis_docs_creating_a_new_shapefile_layer	qgis_docs_using_raster_layers ✓
...	...	...

Note: The underline and checkmark (✓) after the API document title indicates that the document is ground-truth.

6.4 | Can QA4GIS be used on other data sets?

Normally, it cannot be used in other domains. First, our data set is all the API documents about GIS software, so QA4GIS is designed for its content which contains GIS terms, code, and text. In other domains, the questions and answers may contain quite different text. For example, in the physics section of Stack Exchange, many questions and answers contain mathematical formulas which are hard for machine learning models to learn the representations. Second, we added GIS-related prior knowledge to our model using weighted sentence embeddings (detailed in Section 4.3), so the data sets that can be used in QA4GIS should also have a piece of prior knowledge information. Third, a common trend in GIS question answering is to use Knowledge Graph (KG) to answer spatial or geographic questions, for example, the KG data set DB18 mentioned in Mai et al. (2019) and DBGeo mentioned in Mai et al. (2020). But our model is not designed for KGs. We tried unsuccessfully to find an public natural language question answering data set in the GIS domain.

7 | CONCLUSIONS

In this article we did an empirical analysis of questions and answers on the GIS section of the Stack Exchange website and demonstrated the feasibility of answering programming related questions with API documents. We then proposed QA4GIS, a deep learning-based question answering tool containing a DNN model for semantic representation of question–API document pairs, a candidate documents generator, and a learning-to-rank model for ranking the candidates. Our experiments showed that QA4GIS improves 21.39% on the MAP score and 22.34% on the MRR score compared with the best baseline SIF. Finally, we discussed the advantages and limitations of QA4GIS. We also described some applications that we would like to implement as future work.

DATA AVAILABILITY STATEMENT

The data that support the findings of this study are available from the corresponding author upon reasonable request.

ORCID

Xinyue Ye https://orcid.org/0000-0001-8838-9476

REFERENCES

Arora, S., Liang, Y., & Ma, T. (2017). A simple but tough-to-beat baseline for sentence embeddings. In *Proceedings of the International Conference on Learning Representations*, Toulon, France (pp. 1–16).

Bennett, B., Mallenby, D., & Third, A. (2008). An ontology for grounding vague geographic terms. In C. Eschenbach & M. Grüninger (Eds.), *Formal ontology in information systems* (pp. 280–293). Amsterdam, The Netherlands: IOS Press.

Bridle, J. S. (1990). Probabilistic interpretation of feedforward classification network outputs, with relationships to statistical pattern recognition. In F. F. Soulié & J. Hérault (Eds.), *Neurocomputing* (NATO ASI Series F: Computer and Systems Sciences, Vol. *68*, pp. 227–236). Berlin, Germany: 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*. Preprint, arXiv:1608.03905.

Burges, C. J., Ragno, R., & Le, Q. V. (2007). Learning to rank with nonsmooth cost functions. In B. Schölkopf, J. Platt, & T. Hofmann (Eds.), *Advances in Neural Information Processing Systems 19: Proceedings of the 2006 Conference* (pp. 193–200). Cambridge, MA: MIT Press.

Burges, C., Shaked, T., Renshaw, E., Lazier, A., Deeds, M., Hamilton, N., & Hullender, G. (2005). Learning to rank using gradient descent. In *Proceedings of the 22nd International Conference on Machine Learning*, Bonn, Germany (pp. 89–96). New York, NY: ACM.

Cao, Y., Xu, J., Liu, T. Y., Li, H., Huang, Y., & Hon, H. W. (2006). Adapting ranking SVM to document retrieval. In *Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, Seattle, WA (pp. 186–193). New York, NY: ACM.

Cao, Z., Qin, T., Liu, T. Y., Tsai, M. F., & Li, H. (2007). Learning to rank: From pairwise approach to listwise approach. In *Proceedings of the 24th International Conference on Machine Learning*, Corvalis, OR (pp. 129–136). New York, NY: ACM.

Chen, W., Liu, T. Y., Lan, Y., Ma, Z. M., & Li, H. (2009). Ranking measures and loss functions in learning to rank. In Y. Bengio, D. Schuurmans, J. Lafferty, C. Williams, & A. Culotta (Eds.), *Advances in Neural Information Processing Systems 22 (NIPS 2009)* (pp. 315–323). San Diego, CA: Neural Information Processing Systems Foundation.

Cliché, M. (2017). *BB_twtr at SemEval-2017 task 4: Twitter sentiment analysis with CNNs and LSTMs*. Preprint, arXiv:1704.06125

Cohen, D., Mitra, B., Hofmann, K., & Croft, W. B. (2018). Cross domain regularization for neural ranking models using adversarial learning. In *Proceedings of the 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*, Ann Arbor, MI (pp. 1025–1028). New York, NY: ACM.

Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K., & Kuksa, P. (2011). Natural language processing (almost) from scratch. *Journal of Machine Learning Research*, *12*, 2493–2537. https://dl.acm.org/doi/10.5555/1953048.2078186

Cossock, D., & Zhang, T. (2006). Subset ranking using regression. In G. Lugosi & H. U. Simon (Eds.), *Learning theory: COLT 2006* (Lecture Notes in Computer Science, Vol. *4005*, pp. 605–619). Berlin, Germany: Springer.

Crammer, K., & Singer, Y. (2002). Pranking with ranking. In T. Dietterich, S. Becker, & Z. Ghahramani (Eds.), *Advances in Neural Information Processing Systems 14 (NIPS 2001)* (pp. 641–647). San Diego, CA: Neural Information Processing Systems Foundation.

Dai, Z., Xiong, C., Callan, J., & Liu, Z. (2018). Convolutional neural networks for soft-matching n-grams in ad-hoc search. In *Proceedings of the 11th ACM International Conference on Web Search and Data Mining*, Los Angeles, CA (pp. 126–134). New York, NY: ACM.

Dehghani, M., Zamani, H., Severyn, A., Kamps, J., & Croft, W. B. (2017). Neural ranking models with weak supervision. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*, Tokyo, Japan (pp. 65–74). New York, NY: ACM.

Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2018). *BERT: Pre-training of deep bidirectional transformers for language understanding*. Preprint, arXiv:1810.04805.

Dwarampudi, M., & Subba Reddy, N. V. (2019). *Effects of padding on LSTMs and CNNs*. Preprint, arXiv:1903.07288.

Elbassuoni, S., Ramanath, M., & Weikum, G. (2011). Query relaxation for entity-relationship search. In G. Antoniou, M. Grobelnik, E. Simperl, B. Parsia, D. Plexousakis, P. de Leenheer, & J. Z. Pan (Eds.), *The Semantic Web: Research and applications*, *ESWC 2011* (Lecture Notes in Computer Science, Vol. *6644*, pp. 62–76). Berlin, Germany: Springer.

Freund, Y., Iyer, R., Schapire, R. E., & Singer, Y. (2003). An efficient boosting algorithm for combining preferences. *Journal of Machine Learning Research*, *4*, 933–969. https://dl.acm.org/doi/10.5555/945365.964285

Gao, S., & Goodchild, M. F. (2013). Asking spatial questions to identify GIS functionality. In *Proceedings of the Fourth International Conference on Computing for Geospatial Research and Application*, San Jose, CA (pp. 106–110). Piscataway, NJ: IEEE.

Guo, J., Fan, Y., Ai, Q., & Croft, W. B. (2016). Semantic matching by non-linear word transportation for information retrieval. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*, Indianapolis, IN (pp. 701–710). New York, NY: ACM.

Hamilton, W., Bajaj, P., Zitnik, M., Jurafsky, D., & Leskovec, J. (2018). Embedding logical queries on knowledge graphs. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, & R. Garnett (Eds.), *Advances in Neural Information Processing Systems 31 (NeurIPS 2018)* (pp. 2026–2037). San Diego, CA: Neural Information Processing Systems Foundation.

Hamzei, E., Li, H., Vasardani, M., Baldwin, T., Winter, S., & Tomko, M. (2019). Place questions and human-generated answers: A data analysis approach. In P. Kyriakidis, D. Hadjimitsis, D. Skarlatos, & A. Mansourian (Eds.), *Geospatial technologies for local and regional development: Proceedings of the 22nd AGILE Conference on Geographic Information Science* (pp. 3–19). Cham, Switzerland: Springer.

He, H., Ning, Q., & Roth, D. (2019). *QuASE: Question-answer driven sentence encoding*. Preprint, arXiv:1909.00333.

Herbrich, R., Graepel, T., & Obermayer, K. (2000). Large margin rank boundaries for ordinal regression. In A. J. Smola, P. L. Bartlett, B. Schölkopf, & D. Schuurmans (Eds.), *Advances in large margin classifiers* (pp. 115–132). Cambridge, MA: MIT Press.

Jansen, B. J., & Rieh, S. Y. (2010). The seventeen theoretical constructs of information searching and information retrieval. *Journal of the American Society for Information Science and Technology*, *61*(8), 1517–1534. https://doi.org/10.1002/asi.21358

Jeon, J., Croft, W. B., & Lee, J. H. (2005). Finding similar questions in large question and answer archives. In *Proceedings of the 14th ACM International Conference on Information and Knowledge Management*, Bremen, Germany (pp. 84–90). New York, NY: ACM.

Johnson, J., Douze, M., & Jégou, H. (2017). *Billion-scale similarity search with GPU*. Preprint, arXiv:1702:08734.

Karpukhin, V., Oğuz, B., Min, S., Wu, L., Edunov, S., Chen, D., … Yih, W.-T. (2020). *Dense passage retrieval for open-domain question answering*. Preprint, arXiv:2004.04906.

Li, H. (2011a). A short introduction to learning to rank. *IEICE Transactions on Information and Systems*, *94*(10), 1854–1862. https://doi.org/10.1587/transinf.E94.D.1854

Li, H. (2011b). Learning to rank for information retrieval and natural language processing. *Synthesis Lectures on Human Language Technologies*, *4*(1), 1–113. https://doi.org/10.2200/S00348ED1V01Y201104HLT012

Li, J., Sun, A., & Xing, Z. (2018). Learning to answer programming questions with software documentation through social context embedding. *Information Sciences*, *448*, 36–52. https://doi.org/10.1016/j.ins.2018.03.014

Li, W., Song, M., & Tian, Y. (2019). An ontology-driven cyberinfrastructure for intelligent spatiotemporal question answering and open knowledge discovery. *ISPRS International Journal of Geo-Information*, *8*(11), 496. https://doi.org/10.3390/ijgi8110496

MacAvaney, S., Yates, A., Cohan, A., & Goharian, N. (2019). CEDR: Contextualized embeddings for document ranking. In *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, Paris, France (pp. 1101–1104). New York, NY: ACM.

Mai, G., Janowicz, K., Cai, L., Zhu, R., Regalia, B., Yan, B., … Lao, N. (2020). SE-KGE: A location-aware knowledge graph embedding model for geographic question answering and spatial semantic lifting. *Transactions in GIS*, *24*(3), 623–655. https://doi.org/10.1111/tgis.12629

Mai, G., Yan, B., Janowicz, K., & Zhu, R. (2019). Relaxing unanswerable geographic questions using a spatially explicit knowledge graph embedding model. In P. Kyriakidis, D. Hadjimitsis, D. Skarlatos, & A. Mansourian (Eds.), *Geospatial technologies for local and regional development: Proceedings of the 22nd AGILE Conference on Geographic Information Science* (pp. 21–39). Cham, Switzerland: Springer.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., & Dean, J. (2013). Distributed representations of words and phrases and their compositionality. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, & K. Q. Weinberger (Eds.), *Advances in Neural Information Processing Systems 26 (NIPS 2013)* (pp. 3111–3119). San Diego, CA: Neural Information Processing Systems Foundation.

Mitra, B., & Craswell, N. (2018). An introduction to neural information retrieval. *Foundations and Trends in Information Retrieval*, *13*(1), 1–126. https://doi.org/10.1561/1500000061

Mitra, B., Diaz, F., & Craswell, N. (2017). Learning to match using local and distributed representations of text for web search. In *Proceedings of the 26th International Conference on World Wide Web*, Perth, Australia (pp. 1291–1299). New York, NY: ACM.

Nassif, H., Mohtarami, M., & Glass, J. (2016). Learning semantic relatedness in community question answering using neural models. In *Proceedings of the First Workshop on Representation Learning for NLP*, Berlin, Germany (pp. 137–147). Stroudsburg, PA: ACL.

Nogueira, R., & Cho, K. (2017). *Task-oriented query reformulation with reinforcement learning*. Preprint, arXiv:1704.04572.

Pennington, J., Socher, R., & Manning, C. D. (2014). GLoVe: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*, Doha, Qatar (pp. 1532–1543). Stroudsburg, PA: ACL.

Peters, M. E., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K., & Zettlemoyer, L. (2018). Deep contextualized word representations. In *Proceedings of the 16th Annual Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, New Orleans, LA (pp. 2227–2237). Stroudsburg, PA: ACL.

Qiu, X., Sun, T., Xu, Y., Shao, Y., Dai, N., & Huang, X. (2020). *Pre-trained models for natural language processing: A survey*. Preprint, arXiv:2003.08271.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., & Sutskever, I. (2019). Language models are unsupervised multitask learners. *OpenAI Blog*, *1*(8), 9.

Rajpurkar, P., Zhang, J., Lopyrev, K., & Liang, P. (2016). *SQuAD: 100,000+ questions for machine comprehension of text*. Preprint, arXiv:1606.05250.

Robertson, S. E., & Walker, S. (1994). Some simple effective approximations to the 2–Poisson model for probabilistic weighted retrieval. In B. W. Croft & C. J. van Rijsbergen (Eds.), *SIGIR'94: Proceedings of the 17th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (pp. 232–241). London, UK: Springer.

Robertson, S., & Zaragoza, H. (2009). *The probabilistic relevance framework: BM25 and beyond*. Delft, the Netherlands: Now Publishers.

Rosset, C., Jose, D., Ghosh, G., Mitra, B., & Tiwary, S. (2018). Optimizing query evaluations using reinforcement learning for web search. In *Proceedings of the 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*, Ann Arbor, MI (pp. 1193–1196). New York, NY: ACM.

Sanh, V., Debut, L., Chaumond, J., & Wolf, T. (2019). *DistilBERT, a distilled version of BERT: Smaller, faster, cheaper and lighter*. Preprint, arXiv:1910.01108.

Sarvi, F., Voskarides, N., Mooiman, L., Schelter, S., & de Rijke, M. (2020). *A comparison of supervised learning to match methods for product search*. Preprint, arXiv:2007.10296.

Scheider, S., Ballatore, A., & Lemmens, R. (2019). Finding and sharing GIS methods based on the questions they answer. *International Journal of Digital Earth*, *12*(5), 594–613. https://doi.org/10.1080/17538947.2018.1470688

Scheider, S., Meerlo, R., Kasalica, V., & Lamprecht, A. L. (2020). Ontology of core concept data types for answering geo-analytical questions. *Journal of Spatial Information Science*, *20*, 167–201. https://doi.org/10.5311/JOSIS.2020.20.555

Scheider, S., Nyamsuren, E., Kruiger, H., & Xu, H. (2020). Geo-analytical question-answering with GIS. *International Journal of Digital Earth*, *14*(1), 1–14. https://doi.org/10.1080/17538947.2020.1738568

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*, Santiago, Chile (pp. 373–382). New York, NY: ACM.

Severyn, A., & Moschitti, A. (2016). *Modeling relational information in question-answer pairs with convolutional neural networks*. Preprint, arXiv:1604.01178.

Shashua, A., & Levin, A. (2003). Ranking with large margin principle: Two approaches. In S. Becker, S. Thrun, & K. Obermayer (Eds.), *Advances in Neural Information Processing Systems 16: Proceedings of the 2002 Conference* (pp. 961–968). Cambridge, MA: MIT Press.

Taylor, M., Guiver, J., Robertson, S., & Minka, T. (2008). Softrank: Optimizing non-smooth rank metrics. In *Proceedings of the 2008 International Conference on Web Search and Data Mining*, Banff, Alberta, Canada (pp. 77–86). New York, NY: ACM.

Turian, J., Ratinov, L., & Bengio, Y. (2010). Word representations: A simple and general method for semi-supervised learning. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, Uppsala, Sweden (pp. 384–394). Stroudsburg, PA: ACL.

Wu, H. C., Luk, R. W. P., Wong, K. F., & Kwok, K. L. (2008). Interpreting tf-idf term weights as making relevance decisions. *ACM Transactions on Information Systems*, *26*(3), 1–37. https://doi.org/10.1145/1361684.1361686

Wu, Q., Burges, C. J., Svore, K. M., & Gao, J. (2010). Adapting boosting for information retrieval measures. *Information Retrieval*, *13*(3), 254–270. https://doi.org/10.1007/s10791-009-9112-1

Xia, F., Liu, T. Y., Wang, J., Zhang, W., & Li, H. (2008). Listwise approach to learning to rank: Theory and algorithm. In *Proceedings of the 25th International Conference on Machine Learning*, Helsinki, Finland (pp. 1192–1199). New York, NY: ACM.

Xiong, C., Dai, Z., Callan, J., Liu, Z., & Power, R. (2017). End-to-end neural ad-hoc ranking with kernel pooling. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*, Tokyo, Japan (pp. 55–64). New York, NY: ACM.

Xu, J., & Li, H. (2007). Adarank: A boosting algorithm for information retrieval. In *Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, Amsterdam, the Netherlands (pp. 391–398). New York, NY: ACM.

Yang, Z., Qi, P., Zhang, S., Bengio, Y., Cohen, W. W., Salakhutdinov, R., & Manning, C. D. (2018). *HotpotQA: A dataset for diverse, explainable multi-hop question answering*. Preprint, arXiv:1809.09600.

Yue, Y., Finley, T., Radlinski, F., & Joachims, T. (2007). A support vector method for optimizing average precision. In *Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, Amsterdam, the Netherlands (pp. 271–278). New York, NY: ACM.

Zheng, Z., Zha, H., Zhang, T., Chapelle, O., Chen, K., & Sun, G. (2008). A general boosting method and its application to learning ranking functions for web search. In J. C. Platt, D. Koller, Y. Singer, & S. T. Roweis (Eds.), *NIPS'07: Proceedings of the 20th International Conference on Neural Information Processing Systems* (pp. 1697–1704). Red Hook, NY: Curran Associates.

How to cite this article: Wang, W., Li, Y., Wang, S., & Ye, X. (2021). QA4GIS: A novel approach learning to answer GIS developer questions with API documentation. *Transactions in GIS*, 25, 2675–2700. https://doi.org/10.1111/tgis.12798

APPENDIX A

Acronyms and abbreviations

TABLE A1 Acronyms and abbreviations in this paper

Acronym/abbreviation	Description
API	Application Programming Interface
BERT	Bidirectional Encoder Representations from Transformers
CQA	Community-based Question Answering
DNN	Deep Neural Network
ELMo	Embeddings from Language Models
ELU	Exponential Linear Unit
GIS	Geographic Information System
GPT-2	Generative Pretrained Transformer 2
IDF	Inverse Document Frequency
KG	Knowledge Graph
LTR	Learning To Rank
MAP	Mean Average Precision
MRR	Mean Reciprocal Rank
QA	Query-API document
WMD	Word Mover's Distance