

Know-How in Programming Tasks: From Textual Tutorials to Task-Oriented Knowledge Graph

Jiamou Sun*, Zhenchang Xing*, Rui Chu*, Helai Bai*, Jinshui Wang[†], Xin Peng[‡]

*Research School of Computer Science, CECS, Australian National University, Canberra, Australia

[†]School of Information Science and Engineering, Fujian University of Technology, Fuzhou, China

[‡]School of Computer Science, Fudan University, Shanghai, China

{u5871153, Zhenchang.Xing, u6013698, u6341820}@anu.edu.au

ymkscom@gmail.com, {pengxin,17212010022}@fudan.edu.cn

Abstract—Accomplishing a program task usually involves performing multiple activities in a logical order. Task-solving activities may have different relationships, such as subactivity-of, precede-follow, and different attributes, such as location, condition, API, code. We refer to task-solving activities and their relationships and attributes as know-how knowledge. Programming task know-how knowledge is commonly documented in semi-structured textual tutorials. A formative study of the 20 top-viewed Android-tagged how-to questions on Stack Overflow suggests that developers are faced with three information barriers (incoherent modeling of task intent, tutorial information overload and unstructured task activity description) for effectively discovering and understanding task-solving knowledge in textual tutorials. Knowledge graphs have been shown to be effective in representing relational knowledge and supporting knowledge search in a structured way. Unfortunately, existing knowledge graphs extract only know-what information (e.g., APIs, API caveats and API dependencies) from software documentation. In this paper, we devise open information extraction (OpenIE) techniques to extract candidates for task activities, activity attributes and activity relationships from programming task tutorials. The resulting knowledge graph, TaskKG, includes a hierarchical taxonomy of activities, three types of activities relationships, five types of activity attributes, and enables activity-centric knowledge search. As a proof-of-concept, we apply our approach to Android Developer Guide. A comprehensive evaluation of TaskKG shows high accuracy of our OpenIE techniques. A user study shows that TaskKG is promising in helping developers finding correct answers to programming how-to questions.

I. INTRODUCTION

Programming tasks can be tool oriented (e.g., Create app icons with Image Asset Studio in Fig. 1(a)), code oriented (e.g., Use fonts in XML layouts in Fig. 1(b)), and the combination of both (e.g., Start Another Activity in Fig. 1(c)). In these examples, we can observe several common characteristics of programming tasks. First, tasks can be broken down into a *hierarchical taxonomy* of activities from composite activities to atomic actions. For example, “Using fonts in XML layouts” $\subset$ “Adding fonts to a TextView” $\subset$ “set fontFamily attribute” in Fig 1(b). Second, task-solving activities often need to be performed in a temporal order, such as 1) “select Notification Icons” $\rightarrow$ 2) “Select an Asset Type” in Fig 1(a). Third, task-solving activities may involve many attributes. For example, the step “Open the Properties window to set the font for the TextView” in Fig. 1(b) involves the API *TextView* and has the goal “set a font for TextView”.

[†] Jinshui Wang is corresponding author.

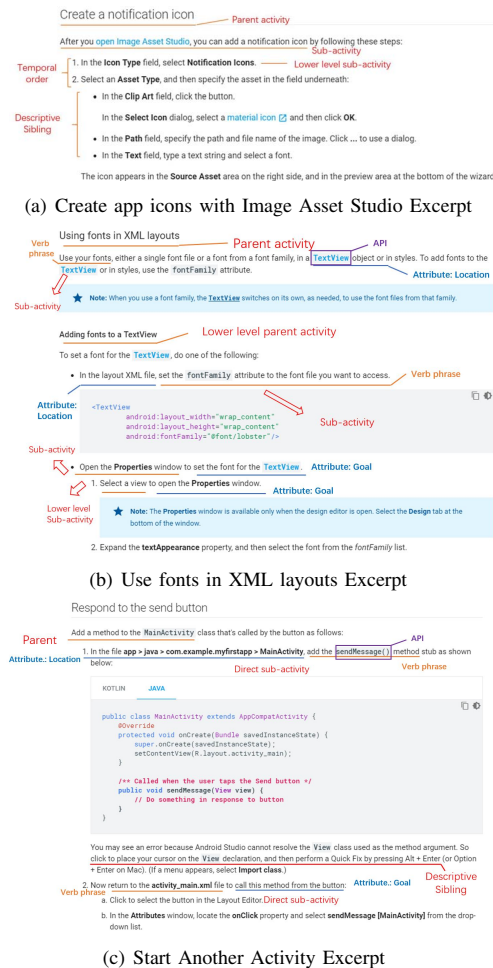


Fig. 1: Examples of Semi-structured Textual Tutorials

In this paper, we refer to task-solving activities, activity attributes and activity relationships as know-how knowledge for programming tasks. Programming task know-how knowledge is commonly documented in semi-structural textual tutorials as examples in Fig. 1. In such tutorials, activity hierarchy and temporal order are implicit in nested headings and numbered lists. Furthermore, most activities and their attributes are buried in unstructured textual descriptions. We hypothesize that such semi-structure textual representation of programming task know-how knowledge is not effective for easy discovery

and understanding of complex know-how knowledge.

To validate our hypothesis, we investigate 20 top-viewed, Android-tagged, how-to questions on Stack Overflow (see Table I). These questions have been viewed in total 13,503,320 times and received 10,329 votes (as of 25th March, 2019). These statistics suggest that developers frequently encounter similar problems and require the task-solving knowledge discussed in the questions and answers. Among these 20 questions, our analysis confirms that seven questions have top-voted answers that are just an excerpt (sometimes paraphrased) from relevant programming tutorials on <https://developer.android.com/>, and nine questions have top-voted answers that contain tutorial excerpts plus some additional explanation provided by the answerers. This observation suggests that Android official tutorials do contain relevant know-how knowledge for many how-to questions that developers frequently have. However, we identify three barriers (incoherent modeling of task intent, tutorial information overload, unstructured task activity description) make it difficult to discover and understand relevant tutorials and relevant knowledge content in tutorials.

To make programming task know-how knowledge more discoverable and easier to understand, we propose to represent task activities, activity relationships and activity attributes as the first-class objects in a task-oriented knowledge graph, rather than as implicit information in document structure and textual description. We parse structured document content to infer composite activities, activity hierarchy and activity temporal order. In addition to composite activities, we train a sentence classifier to extract activity sentences in textual descriptions. We define sentence patterns based on keywords, sentence annotation, and Part-of-Speech (POS) tags to extract action, object and several attributes in activity sentences. As a result, a task-oriented knowledge graph (TaskKG) is constructed. Each activity in the knowledge graph is represented as a structured activity frame including a verb phrase describing the activity, action and object in the verb phrase, links to parent activity, sub-activities, sibling, preceding and following activities, and values of activity attributes.

Based on the task-oriented knowledge graph, we develop an activity-centric knowledge search engine. Given a how-to question, the search engine models the question intent as action, object and activity attributes, as opposed to a bag-of-word. It then finds activity entities in the knowledge graph (as opposed to tutorial documents or fragments) that are most relevant to the detected action, object and attributes in the question. Finally, it composes task-solving answer snippets based on relevant activities. These snippets present programming task know-how knowledge in well-structured activity frames (see Fig. 2), as opposed to unstructured textual descriptions.

As a proof-of-concept, we apply our approach to Android Developer Guide. The resulting TaskKG contains 950 top level activities, 7,722 intermediate non-leaf composite activities, 11,519 atomic leaf actions, 20,191 hierarchical relationships, 2,870 precede-follow relationships, 4,524 descriptive-sibling relationships, and 17,012 attributes. We use statistical

sampling methods [1] to examine the extracted activities, relationships and attributes. At 95% confidence level and 5% error margin, our information extraction methods achieve 0.97 F1-score for activity sentence extraction, 91.2% accuracy for activity frame extraction from activity sentences, 87.35% average accuracy for attribute extraction, and 89.54% average accuracy for relationship extraction. We conduct a user study involving 10 participants and eight Android-related how-to questions. The study results show that participants using our activity-centric knowledge search engine can correctly identify relevant task-solving knowledge with correctness on par with, but 1.76 times faster than those using Google search engine.

This paper makes the following contributions:

- We conduct a formative study of how-to questions on Stack Overflow to investigate information barriers for discovering and understanding programming task know-how knowledge in textual tutorials.
- We develop open information extraction methods to convert semi-structured and unstructured know-how knowledge in textual tutorials into a task-oriented knowledge graph, and develop an activity-centric search engine for answering programming how-to questions.
- We implement a proof-of-concept tool using Android Developer Guide and conduct experiments to evaluate the quality of the constructed knowledge graph and the usefulness of activity-centric knowledge search. Our replication package can be found at https://github.com/sjmsjmdsg/TaskKG_Exp.git.

II. FORMATIVE STUDY

To understand the challenges in discovering and understanding programming task know-how knowledge in textual tutorial, we conduct a formative study of Stack Overflow questions and answers. In this study, we examine Android-tagged questions and answers because Android has comprehensive and well-maintained official API documentation on the Android Developer website, and Android is the most popular mobile platform discussed on Stack Overflow. We examine the questions in a descending order of their view counts. The rationale is that top-viewed questions generally represent the problems that developers frequently encounter in their work. As we are interested in know-how knowledge, we select the questions whose title is “How to ...”.

Two authors examine the top 20 qualified questions according to our selection criteria. We collect question view count and vote statistics to estimate the potential impact of these questions on developers. We analyze the question descriptions to confirm if the askers attempted to solve the task but failed, or just simply asked how to solve the task. Another key analysis is to examine whether the top-voted answers to these 20 how-to questions contain task know-how knowledge from official Android tutorials (excluding API reference documentation) on the Android Developer website. Some answers contain hyperlinks to the source of answer content which facilitate this analysis. We also use Google to search the Android Developer website using question titles

TABLE I: Statistic of 20 top-viewed, Android-tagged, how-to questions on Stack Overflow (A.E-Additional Explanation; DirExcept-Direct Excerpt; StL-Step Level, xSt-x Steps, SuL-Subsection Level, xSu-x Subsections, xAW-x Average Words, Atte-Attempted)

Questions	Atte	DirExcept or Paraphrased	A.E	Link to Tutorial	Complexity of Tutorial
1.How to display Toast in Android?	Yes	DirExcept	No	The Basics	StL, 2St, 48AW
2.How to set the text color of TextView in code?	Yes	No Tutorial			
3.How to Change Android SDK Path tools	No	Paraphrased	No	Update with SDK Manager	SuL, 1Su, 626AW
4.How to hide the title bar for an Activity in XML with existing custom theme	Yes	DirExcept	No	Hide the Status Bar on Android 4.0 and Lower	SuL, 2Su, 268AW
5.How to start new activity on button click	No	Paraphrased	No	Build an Intent	SuL, 4Su, 396AW
6.How to change fontFamily of TextView in Android	Yes	Paraphrased	Yes	Adding fonts to a TextView	StL, 1St, 17AW
7.How to add an image to the drawable folder?	Yes	Paraphrased	Yes	Create action bar or tab icon	SuL, 1Su, 454AW
8.How to make a background 20% transparent on Android	No	Paraphrased	Yes	Color	SuL, 1Su, 281AW
9.How to change background color in android app Themes	No	Paraphrased	Yes	Styles and	SuL, 3Su, 136AW
10.How to add manifest permission to an application?	Yes	DirExcept	No	Connect to the network	StL, 1St, 16AW
11.How to use putExtra() and getExtra() for string data	Yes	Paraphrased	Yes	Start another activity	SuL, 2Su, 341AW
12.How to open the Google Play Store directly from my Android application?	Yes	No Tutorial			
13.How to change the status bar color in android	No	Paraphrased	Yes	Customize the default theme	SuL, 2Su, 521AW
14.How to manage startActivityForResult on Android?	No	Paraphrased	Yes	Start the Activity	SuL, 2Su, 87AW
15.How to use SharedPreferences in Android to store, fetch and edit values	No	DirExcept	Yes	Save key-value data	SuL, 3Su, 198AW
16.How to run a Runnable thread in Android at defined intervals?	Yes	No Tutorial			
17.How to convert a Drawable to a Bitmap?	Yes	Paraphrased	No	Load Bitmaps Efficiently	SuL, 2Su, 180AW
18.How to pass an object from one activity to another on Android	No	Paraphrased	Yes	Start another activity	SuL, 2Su, 341AW
19.How to check certificate name and alias in keystore files?	Yes	No Tutorial			
20.How to send an object from one Android Activity to another using Intents?	Yes	Paraphrased	No	Parcelables and Bundles	SuL, 2Su, 229AW

and keywords in answers. Once we identify some official Android tutorials as the knowledge source of an answer, we compare the answer content with the tutorial content to see if the answer content is just an excerpt of tutorial content, or paraphrased tutorial content, or tutorial content enriched with additional answerer explanation. We also identify whether the tutorial excerpt is at the subsection level or specific step level, how many subsection(s) or step(s) are involved, and how many words in these subsections or steps. Based on our search experience and tutorial statistics, we attempt to identify the information barriers in discovering and understanding task know-how knowledge in textual tutorials. One author performs the analysis and the other author validates the analysis results. If there are any disagreements, the controversy will be discussed by all co-authors to reach the consensus.

Table I presents our analysis results. We find that the askers of eight questions attempted to solve the tasks but failed, and then came to Stack Overflow to seek help, even though official Android tutorials contain relevant task-solving knowledge. According to our analysis, we do not find relevant tutorials on the Android Developer website for only four questions (Q2, Q12, Q16, and Q19). Many approaches [2]–[4] have been proposed to exploit crowd documentation (e.g., Stack Overflow questions and answers) to supplement official documentation. In this work, we focus on the remaining 16 questions, among which seven questions have top-voted answers that are just an excerpt (sometimes paraphrased) of relevant tutorial content on the Android Developer, and nine questions have top-voted answers that contain tutorial excerpts with some additional explanation provided by the answerers.

Our analysis of these 18 questions reveals three information barriers that make it difficult for developers to discover and understand task solving knowledge in textual tutorials.

Incoherent modeling of task intent: As seen in Table I, the intent of a how-to question is a verb phrase, comprising coherent parts such as action, object and some attributes, for example “hide” (action), “title bar” (object) and “in XML” (location) in Question#4. Task activity description in tutorials share the same linguistic characteristics (see Fig. 1). Modeling a task question as a bag of words loses the coherence of task intent, and as a result it may retrieve textually similar but activity wise irrelevant tutorials, especially when the task question is general like “change background color” (Question#9) or long like “hide title bar for an Activity in XML with existing custom theme” (Question#4). Searching “change background color” on the Android Developer website finds as the top-1 result The Color System. But this document just introduces the Android color system but does not describe how to change background color. Searching the long question title of Question#4 on the Android Developer website finds Styles and Themes, which has nothing to do with “hide title bar in XML”. To overcome the incoherent modeling of task intent, we need to devise a coherent activity-centric representation of task questions and activity descriptions and support activity-centric intent comparison and matching.

Tutorial information overload: Among the 16 questions that have relevant tutorials on the Android Developer website, searching the website with the question titles can find the most relevant tutorials at the top-1 rank for 11 questions (e.g., Question#5/6/7/14/15). However, the relevant tutorials usually contain a large volume of information. Consider Question#6. Searching “change fontFamily of TextView” returns the relevant tutorial Fonts in XML. This tutorial has 5 top-level subsections, each of which is a font-related task. Each task have several steps with code examples and screenshots. However, the solution to Question#6 involves one key step

with 17 words buried in all this overwhelming information. Such information overload makes it difficult to find the most relevant task-solving knowledge in the tutorial for “change fontfamily of TextView”. To overcome the tutorial information overload issue, we need activity-centric knowledge search, rather than document-level information retrieval.

Unstructured task activity description: As the tutorial examples in Fig 1 show, although coarse-grained composite activities are expressed in structured content (e.g., nested headings), fine-grained task activities are described in unstructured text. A paragraph of unstructured text may involve several fine-grained activities, each of which may involve some APIs, conditions, goals, and/or locations. Such convoluted, unstructured task activity descriptions may cause misunderstanding or overlooking of relevant task-solving knowledge, which in turn results in errors in how-to questions (e.g., Question#10). To support intuitive understanding of task activities, we need to convert task-solving knowledge from implicit information in semi-structured or unstructured text into explicit task activities, activity attributes and activity relationships.

III. APPROACH

We propose a knowledge graph based approach to overcome the three information barriers revealed in our formative study. As shown in Fig. 2, our knowledge graph represents programming task know-how knowledge as explicit task activities, activity attributes and activity relationships, and encapsulates activity information structurally in activity frames (Section III-A). We devise open information extract (OpenIE) methods to construct a task-oriented knowledge graph (TaskKG) from semi-structured textual tutorials (Section III-B). The resulting TaskKG supports activity-centric query intent modeling, candidate activity matching and task know-how knowledge presentation (Section III-C).

A. Explicit Notion of Task Know-How Knowledge

Task activity: We view a programming task as a hierarchy of activities. A *non-leaf activity* can be decomposed into a set of sub-activities. A *leaf activity* is an atomic action.

Activity attribute: We consider five types of activity attributes: 1) *API* involved in an activity (e.g., `TextView` in the activity “set the font for `TextView`”); 2) *location* where an activity occurs (e.g., *in XML file, within Android Studio, from the drop-down list*); 3) *condition* to satisfy for performing an activity (e.g., *if you use a new resource qualifier*); 4) *goal* that an activity achieves (e.g., *to preview the UI, so that the highlights around them are clearly visible*); 5) *code snippet(s)* demonstrating one or several activities.

Activity relationships We consider three types of activity relationships: 1) *hierarchal relationship* between a parent activity and its sub-activities; 2) *descriptive sibling* between a sibling activity mentioned immediately before another sibling activity (e.g., “click to place your cursor on the View declaration” before “perform a Quick Fix by pressing Alt + Enter” in Fig. 1(c)). 3) *temporal relationship* between a preceding sibling activity and a following sibling activity with clear step-indicators (e.g., the step 1 “select Notification Icons” precedes the step 2 “Select an Asset Type” in Fig. 1(a)). The

temporal relationship (if present between two sibling activities) overrides the descriptive sibling.

Task activity frame: An activity frame is a structured representation of the information of a task activity. It has three compartments: 1) *verb phrase* including an action and an object; 2) *attributes* (API, location, condition, goal, and code); and 3) *relationships* (references to parent activity, sub-activities, preceding and following activities, and/or descriptive-before and descriptive-after activities). The verb phrase cannot be NULL, while attributes and relationships can be NULL.

Task-oriented knowledge graph (TaskKG): A task-oriented knowledge graph represents task know-how knowledge in a direct graph $G(T, R)$ where T is a set of activity frames and R is a set of relationships between activity frames.

B. Constructing TaskKG by OpenIE Methods

The input to our approach is semi-structured textual tutorials that document task know-how knowledge for an application domain, for example Android Developer Guide. Based on the structured content of such tutorials and the linguistic characteristics of task descriptions, we devise natural language processing techniques and heuristics to extract task activities, activity attributes and activity relationships from the tutorials, which are encapsulated in activity frames to form a task-oriented knowledge graph.

1) Activity Extraction: As shown in Fig. 1, task tutorials usually describe coarse-grained composite activities in a semi-structured way by document tree and nested headings, and discuss fine-grained activities in text. Coarse-grained composite activities can be easily extracted from the structured content, for example by analyzing document tree (such as `<ul>` of list structures and `<p>` of non-list structures) and heading (such as `<h1>` and `<h2>`) written in markup languages.

The key challenge is to extract fine-grained activities discussed in text. We observe that fine-grained activities in text are usually expressed in imperative sentences. However, not all imperative sentences describe task activities. For example, “increase app installation success rate” is an imperative sentence but not an activity. Table II shows some more examples. Existing methods [5] defines programming action verbs and object nouns to identify activity sentences, and non-programming action verbs and object nouns to filter out non-activity sentences. As indicated in Table II, such pre-defined verb and noun lists are often too rigid to handle the diversity of activity descriptions. For example, “display” is a programming action verb defined in [5], but “the system displays a stream notification” is not an activity sentence. In contrast, “tap” is a non-programing action verb defined in [5], but “tap the Overflow button” is an activity sentence.

Therefore, we develop a sentence classifier to classify whether a sentence is an activity sentence or not. Our classifier is an attention-based Bidirectional Long-Short-Term Memory (Bi-LSTM) network [6], which models bidirectional context of a word. It takes as input the word embedding representation of a sentence. We use the start-of-the-art pre-trained word embeddings BERT [7]. The network uses 128 hidden states

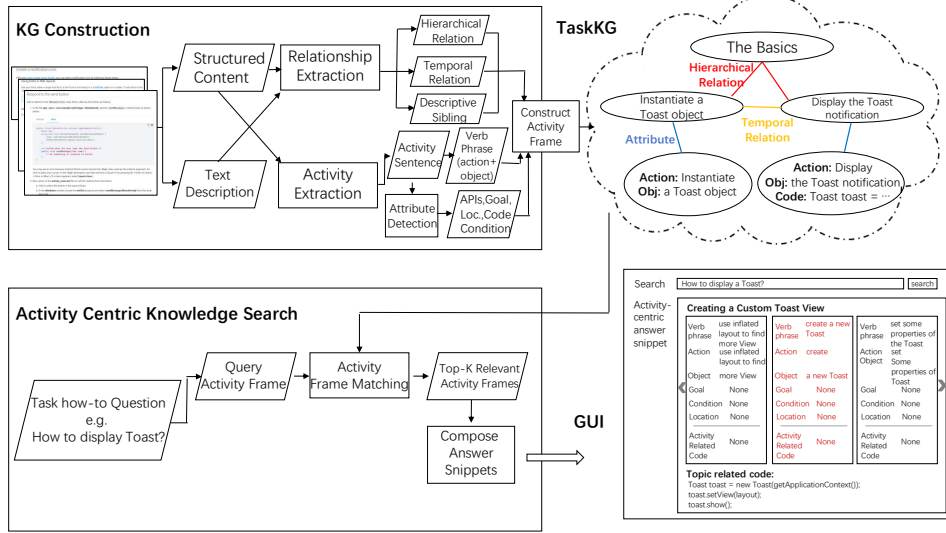


Fig. 2: Approach Overview

TABLE II: Examples of Activity versus Non-Activity Sentences

	Candidate Sentences	TaskNav
Activity	1.Click Start a new Android Studio project	Yes
	2.Add a method to the MainActivity class	Yes
	3.Tap the Overflow button	No
	4.Inflate the string array	No
Non-Activity	1.Now take a moment to review the most important files	No
	2.Take note of the details in this method in the TextView	No
	3.the system displays a stream notification	Yes
	4....which it might make sense to request a permission	Yes

to embed the input sentence. The output sentence vector is fed into a Softmax classifier to classify the sentence.

We then parse the activity sentence to extract the action and object of the sentence. The action can be a verb or verb phrase, and the object can be a noun or noun phrase. We first use Stanford CoreNLP tool [8] to obtain the Part-of-Speech (POS) tags of an activity sentence. Then, we adopt the phrase chunking technique proposed in [9] for constructing software technique knowledge graph from web content. Specifically, we extract verb (or verb phrase) as action and noun (or noun phrase) as an object based on a set of POS patterns designed for parsing software text. Our POS patterns handle software-specific verb phrases, such as “click Start a new Android Studio Project”, which will be chunked into “click” as an action and “Start a new Android Studio Project” as an object.

2) *Attribute Extraction*: For each activity sentence, we extract a set of activity attributes (if any). APIs in a sentence are often annotated with special tags (e.g., `TextView` in “set the font for `TextView` in Fig. 1(b)). We first extract candidate text fragments in the activity sentence by looking for such special tags. If an annotated text fragment is a hyperlink to an API reference document, we extract the annotated text fragment as an API attribute involved in the activity. Furthermore, we use orthographic feature based regular expressions [10]

TABLE III: Keywords for Location/Condition/Goal Attributes

Attribute	Keywords	Example
Condition	if	1. If your editor shows the XML source, click the Design tab at the bottom of the window
Goal	to, in order, so that	1. To access the updater, use the android command-line tool
Location	with, at, on, near, into, inside, in, from, under, within	1.Click-and-hold the anchor on the top side 2.Change to a custom locale from the adb shell

to detect API mentions without special tag annotation, (e.g., `sendMessage()` in the activity sentence “add the `sendMessage()` method stub” in Fig. 1(c)). These regular expressions are also used to detect API mentions in task how-to questions.

Code snippets in text are always annotated with special tags and they can be extracted in the same way as APIs in text. If a code snippet is immediately after an activity sentence (e.g., the code snippet immediately after the step 1 “add the `sendMessage()` method stub” in Fig. 1(c)), we attach the code snippet to this particular activity as activity-related code. Otherwise, we attached the code snippet to the parent composite activity. Usually, such code snippets illustrates several sub-activities of the parent activity as a topic summary. We call such code snippets as topic-related code.

While we label sentences for training activity sentence classifier, we summarize keywords indicating location, condition or goal attributes in the activity sentences. Table III shows the summarized keywords and some typical examples for each attribute type. Given an activity sentence, we look for these keywords in the sentence and extract the relevant phrase as the location, condition or goal attribute of an activity.

3) *Relationship Extraction*: We extract hierarchical relationships from a document tree, nested content headings or nested lists. As illustrated in Fig. 1, the activities at an higher level of content structure are parent activities, and the activities at one-level lower are direct sub-activities of these parent activities,

for example, “Using fonts in XML layouts” $\subset$ “Adding fonts to a TextView” $\subset$ “set the fontFamily attribute to the font file you want to access” in Fig. 1(b).

Direct sub-activities of a parent activity are sibling activities, for example, “add the sendMessage() method stub” and “return to the activity_main.xml file” in Fig. 1(c). In this work, we assume that document authors would organize and explain sibling activities in a logical way, for example, “click to place your cursor on the View declaration”, “perform a Quick Fix by pressing Alt + Enter” and “select Import class” in a paragraph in Fig. 1(c). We use descriptive sibling relationships to capture this natural logical order of sibling activities. In addition, we heuristically infer a specific type of descriptive sibling relationships in this work, i.e., temporal relationships for sibling activities with clear step-like indicators (e.g., the step 1 “select Notification Icons” precedes the step 2 “Select an Asset Type” in Fig. 1(a)). If no such step-like indicators are present (e.g., the · list in Fig. 1(a)) between sibling activities, we consider only general descriptive sibling relationships between these sibling activities.

C. TaskKG Empowered Activity-Centric Knowledge Search

An important usage of knowledge graph is to support entity-centric search with the goal to “find the things, not the strings” [11]. The entities in our task-oriented knowledge graph are activities. We develop an activity-centric search engine which returns relevant activities in the TaskKG for an input task how-to query. The query-activity matching and search results presentation are based on activity frames (a coherent, structured representation of activity information).

1) Query-Activity Matching: Given a task how-to query in natural language description, we first use the same activity and attribute parsing techniques as described in Section III-B to detect action, object and attribute(s) in the query. The detected activity information is formulated as a query activity frame. We then match the query activity frame with the activity frames in the TaskKG by comparing the verb-phrase and attribute compartments of the activity frames. For the verb-phrase compartment, we compare the action phrase and the object phrase respectively. For the attributes compartment, we compare API element, location phrase, condition phrase and goal phrase respectively. The matching results of verb-phrase and attributes compartments are combined to rank the candidate activity frames.

Given a pair of phrase candidates, we represent the words in the phrases using the start-of-the-art pre-trained BERT word embeddings [7], and compute phrase embeddings by average pooling of the word embeddings of the words in the phrases. We then compute the phrase similarity by the cosine similarity of phrase embeddings. If the phrase similarity is above the user-specified threshold (0.8 in this work), we consider the pair of candidate phrases as a match. Using word embedding based phrase representation can mitigate the lexical gap between the query description and the activity description [12], [13].

The task query can involve certain API element, but will not contain a code snippet. However, an activity frame in the TaskKG can have both API elements and code snippet.

Therefore, we try to match the API element mentioned in the query with both the API and code-snippet attributes of the activity frames in the TaskKG. As APIs may not always be mentioned in fully-qualified names, we perform partial keyword matching between API elements. As long as two API mentions are partially matched, we consider these two API mentions as a match. For example, `putExtra` and `putExtra(String, String[])`, `startActivityForResult` and `startActivityForResult(pickContactIntent, PICK_CONTACT_REQUEST)` are considered matching APIs. This partial keyword matching method has been shown to be effective to match API mentions in informal software text [3].

After matching the verb-phrase and attributes compartments of the query activity frame and the activity frames in the TaskKG, we rank the activity frames in the TaskKG by the sum of the similarity of the matched counterparts of activity frames. If the two activity frames have the same similarity score, we further rank them by the sentence-embedding similarity between the how-to query and the activity description from which an activity frame is extracted. The search engine then use the top-k activity frames to compose answer snippets.

2) Composing Activity-Centric Answer Snippets: We scan the top-k ranked activity frames in a descending order and compose an answer snippet for each top-ranked activity as follows. For a top-ranked activity frame, we traverse its parent-of relationship recursively to find the closest ancestor composite activity extracted from structured content (e.g., headings). Furthermore, we traverse the temporal relationships or the descriptive siblings of the top-ranked activity frame recursively to obtain its sibling activities. Ancestor and sibling activities give the context to judge the relevance of the answer snippet to the query and remind the searchers of the necessary sibling activities around the top-ranked activity. We currently consider up-to four sibling activities (if any) surrounding the top-ranked activity frame (as most of activities have four or fewer sibling activities). If two or more top-ranked activities are sibling activities, we will merge their answer snippets as one answer snippet. In such cases, we will reduce the number of contextual sibling activities and make room for multiple top-ranked sibling activities in the same answer snippet.

We design a user interface (UI) to present activity-centric answer snippets for a task how-to query (see Fig. 2). For each answer snippet, the ancestor composite activity is shown as a title. The top-ranked activity frame and its contextual sibling activity frames are shown in structured boxes, which show the verb phrase, action, object and attributes of each activity. The top-ranked activity is shown in red. The activity-related code snippet of an activity is shown inside the box of the activity frame, while topic-related code snippet of the ancestor activity is shown underneath the activity frames. Users can click left- or right-navigation button to see more preceding (or descriptive before) or following (or descriptive after) sibling activities. Our activity-centric answer snippet UI presents task-solving knowledge in a more structured way to support intuitive understanding of task-solving knowledge.

D. Proof-of-Concept Implementation

We apply our knowledge graph construction methods described in Section III-B to Android Developer Guide. The resulting TaskKG consists of 950 top-level tasks, 7,722 intermediate non-leaf activities, and 11,519 atomic actions. These activities have 20,191 hierarchical relationships, 4,524 descriptive-sibling relationships, and 2,870 temporal relationships. We extract various attributes for these activities: 1,681 goals, 1,443 conditions, 7,027 locations, and 4,739 code snippets. We also implement a search tool as described in Section III-C that takes as input a task query, searches activity frames in the TaskKG and outputs activity-centric answer snippets for the query.

IV. QUALITY OF THE CONSTRUCTED TASKKG

First, we want to confirm the accuracy of our information extraction methods to build the confidence in the quality of the resulting task-oriented knowledge graph.

A. Accuracy of Activity Extraction

Activities are extracted from two main sources: structured document content and activity textual description. Extracting activities from structured document content involves parsing of the document tree and heading tags. After careful implementation and verification of this document parsing step, extracting activities from structured document content will not produce erroneous activities. This means we mainly evaluate the activity extraction from text.

1) *Activity Sentence Extraction*: We use an attention BiLSTM based sentence classifier to extract activity sentences from activity textual description. We create a dataset of 29,611 activity sentences and 30,990 non-activity sentences by manual labeling and synonym-based up-sampling [14]. Using this dataset, we conduct 10-fold cross validation to evaluate the precision, recall and F1-score of our sentence classifier. Precision is the percentage of predicted activity sentences being ground-truth activity sentences. Recall is the percentage of ground-truth activity sentences being correctly classified. F1-score is the harmonic average of the precision and recall. We use three methods commonly used for sentence classification in literature as baselines: linear Support Vector Machine (SVM) [15], TextCNN [16] and Transformer [17].

As shown in Table IV, our method achieves 0.96 precision, 0.98 recall and 0.97 F1-score. As neural network based methods like ours, TextCNN and Transformer can extract most informative text features from the input sentences for the classification task, they all statistically significantly outperform linear SVM which directly classifies the input sentences in their raw form. For activity sentence classification, our method combining attention mechanism and bidirectional LSTM achieves the performance on par with the state-of-the-art CNN-based method TextCNN, and achieves slightly better F1-score than attention-only method Transformer.

2) *Activity Frame Extraction*: We manually examine the accuracy of activity frames extracted from activity sentences. Our Android TaskKG contains 13,793 activities extracted from activity sentences. Examining all of these activities require significant time and manual efforts. Therefore, we sample the

TABLE IV: Performance of Activity Sentence Classification

Method	Precision	Recall	F1-score
Our method	0.96	0.98	0.97
Linear SVM	0.60	0.75	0.65
TextCNN	0.97	0.95	0.96
Transformer	0.96	0.91	0.93

minimal number *MIN* of activity frames in the constructed knowledge graph to ensure the estimated accuracy in the samples has the error margin 0.05 at 95% confidence level [1], [18]. For each sampled activity frame, the two authors first independently evaluate the accuracy of its verb phrase (binary decision) by checking the original activity sentence. For the verb phrases that the two authors disagree, they have to discuss and come to a final decision. Based on the final decisions, we compute the accuracy for the extracted activity frames.

According to our analysis, the accuracy of the sampled 374 activity frames is 91.2% accuracy for the error margin 0.05 at 95% confidence level. The main source of errors is the incorrect input activity sentences, which result in meaningless activity frames. For the correct input activity sentence, the accuracy of extracting verb phrase and chunking action and object is 99.0% and 97.3%, respectively. This shows our verb phrase extraction and chunking rules are effective. Errors in chunking action and object typically involve preposition. Our phrase chunking rules involve prepositions because they are useful to chunk verb phrases like “try to get” in which actions do involve prepositions. However, such chunking rules result in some erroneous action/object chunking. For example, “access each of the select files” is chunked into “access each of” as action and “the selected files” as object.

We compare our method with rule-based task phrase extraction method in TaskNav [5]. TaskNav provides an online tool (<http://task-phrases.herokuapp.com/>) for experimenting their task phrase extraction method. We manually enter the original activity sentence from which a sampled activity frame is extracted into the TaskNav online tool and examine the accuracy of its extracted task phrases. TaskNav achieves 42.2% accuracy. TaskNav has low accuracy because the rules it implements are rigid and consider many activity sentence as non-programming related (see examples in Table II and more examples can be found in our replication package). If TaskNav correctly recognizes an activity sentence, it can accurately extract verb phrase in the sentence (98.1%).

B. Attribute Extraction

For fine-grained modeling of activity information, we extract five types of attributes (API, location, condition, goal and code) from activity sentences and descriptions. We have to manually examine the accuracy of the extracted attributes. Again, we use the statistical sampling described in Section IV-A2 for each type of attribute to estimate the accuracy of the extracted attributes for the error margin 0.05 at 95% confidence level. The two authors examine each sampled attribute independently to evaluate its accuracy and they discuss to reach the consensus if their independent judgments are

different.

Our method achieves 89.8%, 82.2%, 92.8%, 89.5% and 82.6% accuracy for extracting API, location, condition, goal and code snippet, respectively. Extracting API, condition and goal is relatively more accurate than extracting location and code snippet. API elements, if enclosed in hyperlinks to API reference documents, can be accurately extracted. However, many mentions of API elements in text are not enclosed in such hyperlinks. We currently use regular expressions of orthographic features (e.g., camel case, bracket) (similar to [10]) to extract such API mentions. These API extraction regular expressions can make errors. Such errors can be reduced by training machine-learning based API extraction model [10].

Location, condition and goal extraction errors are mainly because our current attribute extraction method is based on simple keyword matching. For example, “to get started” will be extracted as a goal, and “if necessary” will be extracted as a condition. But such goals and conditions carry little meaning so we consider them as errors. Location extraction has more errors because we have to define many location keywords (much more than other types of attributes) to handle the diversity of location expressions. However, the more keywords used, the more likely it is to obtain some noisy results. For example, the activity “Use the hasSystemFeature (String) method to check for specific features at runtime” includes keyword “at”, and “at runtime” is erroneously extracted as a location. In the future, we will use more robust a machine-learning based method like the one we design for activity sentence extraction to improve the accuracy of attribute extraction.

Code snippets are always enclosed in special tags. So extracting code snippets from text has no errors. But our heuristics (see Section III-B2) to link an extracted code snippet to the corresponding activity may have errors. We link a code snippet to the activity immediately before the code snippet, but sometimes the most related activity appears after the code snippet. Furthermore, when the code snippet does not appear immediately after an activity, we link the code snippet to the parent composite activity. But when the composite activity has many sub-activities, it can be difficult to determine which specific sub-activities are related to this code snippet. Feature location techniques [19], [20] may be adopted to improve the accuracy of this traceability recovery problem.

C. Relationship Extraction

We extract three types of activity relationships: hierarchical relationships from structured content; descriptive order from the sibling activity mention order in text; and temporal relationships with clear step-like indicators. Descriptive order is determined by sentence order in text and has no extraction errors. We use the sample statistical sampling method described in Section IV-A2 to examine the *MIN* number of parent-of, subactivity-of, preceding-following relationships respectively for the error margin 0.05 at 95% confidence level. Again, the two authors examine each sampled relationship independently to evaluate its accuracy again the relevant parts of the tutorial documents. They discuss to reach the consensus if their independent judgments are different.

TABLE V: Eight Search Tasks in User Study

1.How to display Toast in Android? <i>Set duration to LENGTH_LONG and set text to "This is my Toast message!"</i>	easy
2.How to hide the title bar for an Activity in XML with existing custom theme? <i>Assume the android version is 4.0 and lower</i>	easy
3.How to use SharedPreferences in Android to store, fetch and edit values? <i>Write the process of get edit and apply SharedPreferences</i>	easy
4.How to change fontFamily of TextView in Android? <i>Assume you want to add the fontFamily "font_family_light" to TextView</i>	medium
5.How to change background color in android app? <i>Assume the color name for background is "@color/white" and you have to add name "white" with value "#FFFFFF" to the resources</i>	medium
6.How to add manifest permission to an application? <i>Assume you get error: WARN/System.err(223):java.net.SocketException:Permission denied (maybe missing INTERNET permission)</i>	medium
7.How to start new activity on button click? <i>Assume you have to launch a new activity, and you want to send a message with name "key" and value named value using putExtra</i>	hard
8.How to manage startActivityForResult on Android? <i>Write how you use startActivityForResult() to send the new Intent, and how you justify the returned result is OK within "onActivityResult"</i>	hard

For parent-of, subactivity-of, preceding-following relationships, the accuracy is 91.7%, 88.2% and 88.8%, respectively. The errors are mainly due to the errors of activity sentence extraction. For example, if a parent activity is not correctly extracted, the sub-activities will be mistakenly linked to the ancestor activity. Similarly, if a sub-activity or sibling activity is not correctly extracted, it will result in wrong subactivity-of or preceding-following relationships.

Our activity, attribute and relationship extraction methods for constructing task-oriented knowledge graph are accurate. This ensures the high quality of the resulting task-oriented knowledge graph for downstream applications.

V. USEFULNESS EVALUATION

Next, we want to investigate the usefulness of activity-centric knowledge search that our TaskKG empowers for recommending relevant task-solving knowledge for programming how-to questions, compared with text-based document or paragraph retrieval.

A. User Study Design

Search tasks: We design search tasks based on the 20 how-to questions investigated in our formative study. Because developers frequently encounter similar problems as these questions, the search tasks can simulate the real-world information needs of developers. We select the questions for which developers attempted the programming tasks but failed. Such questions have more specific information needs than those questions that simply request some general task-solving knowledge. Furthermore, we analyze the questions to determine their difficulty levels based on the complexity of programming tasks and the information volume of relevant tutorials and tutorial contents. Finally, we select eight questions listed in Table V as our search tasks, including three easy, three moderate and two hard questions. We analyze each question

and its answers to identify the specific information needs for the search task (explained in italic font for each question). In our formative study, all eight questions are confirmed to have relevant task-solving knowledge in Android Developer Guide (see Table I). We use relevant tutorial excerpts as the ground-truth answers to the search tasks to evaluate the correctness of the participants' answers.

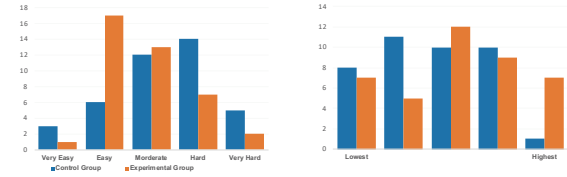
Participants: We recruit 10 senior undergraduate students in our school for the study. Based on the pre-study background survey, six participants have little Android development experience, and the other four have some Android development experience through the school's one-year industry attachment program. However, none of the participants directly know the answers to the eight search tasks. We randomly allocate the participants with comparable Android development experience into five pairs. For each pair of participants, they are randomly assigned to use either our activity-centric search engine or Google to complete the search tasks.

Baseline: We use Google search engine as the baseline tool. Google supports not only the most powerful document retrieval but also paragraph-level results snippets. A result snippet extracts the most relevant paragraph in one of the top search results (the exact mechanism of how Google does this is unknown though), which may directly answer the searcher's question without the need for reading the whole document. We allow the participants to use Google to search the whole Internet and read any search results they want to read, except for the original Stack Overflow questions from which the search tasks are derived. This is to simulate the situation in which no relevant Stack Overflow questions for the search tasks are available, so developers have to find task-solving knowledge in official documentation or other resources.

Experiment procedure: We have an experimental group using our activity-centric search engine and a control group using Google. From the user perspective, our activity-centric search engine works in the same way as other search engines, taking a natural language query and results top ranked search results. We give the experimental group a 10-minute tutorial of the unique concepts in our tool (activity frame, activity attributes, activity relationships, activity/topic-related code snippets). As these concepts are very natural, the participants have no difficulty in understanding them. The participants are given the eight search tasks one at a time, and they complete the task independently. The participants can formulate any query they wish based on the search task descriptions and the hints from previous search results. Each task is given up to seven minutes so that experiment can be controlled within an hour. For each task, the participants are asked to submit an answer with detailed information (not just the reference to relevant documents) for solving the task. They can copy-paste the document content they find relevant in their answers. At the end of each task, they rate the task difficulty and answer confidence in 5-point likert scale. The participants use a screen-recording tool to record their task completion process. Using the task videos, we determine the task completion time and observe the participants' search behavior. After the exper-

TABLE VI: Performance Comparison

Experimental Group		
	Ave Task Time (seconds)	# Correct Answers
P1	166.6	4
P2	147.8	6
P3	144.9	6
P4	141.6	7
P5	158.8	8
Ave±stddev	151.9±9.3	
Control Group		
P6	223.4	3
P7	251.3	5
P8	258.8	7
P9	296.0	7
P10	305.6	8
Ave±stddev	267.0±30.2	



(a) Question Difficulty

(b) Answer Confidence

Fig. 3: Question-Difficulty Ratings and Answer-Confidence Ratings. In the experiment, the two authors collaboratively mark the correctness of the participants' answers against our ground-truth answers.

B. Results and Findings

Table VI shows the average task completion time and the number of correct answers by each participant in the two groups. We can see that the two groups have the similar answer correctness but the experimental group complete the task faster with narrower standard deviation than the control group. Fig. 3 shows the task-difficulty and answer-confidence ratings by the participants in the two groups. We can see that the experimental group participants rate task difficulty as very easy or easy nine times more than the control group participants, and rate answer confidence as high or highest six times more than the control group participants.

By reviewing the task videos, we find that Google can return very relevant online documents for user queries, including not only official Android documentation but also many other relevant resources (e.g., technical blogs for sharing development experience on specific programming tasks). For example, P3 finds the answer to Task#1 in a technical blog "Android Toast Example". Furthermore, Google provides effective results snippets five times for user queries from which the participants can directly locate the most relevant paragraphs in the documents for finding answers. Our activity-centric search engine achieves the performance on par with the powerful Google search engine for finding relevant task-solving knowledge for programming how-to questions.

We also look into the correct answer rate for each search task. Both groups have two correct answers for the two hard search tasks (Task#7/8). That is, neither our search engine nor Google perform well on these two tasks. The experimental group has two more correct answers than the control group for the three moderate search tasks (Task#4/5/6), but they have one more time incorrect answer than the control group for the three easy search tasks (Task#1/2/3). This incorrect answer is

made by P4 on Task#1. He simply copy-paste the activity-related code snippet from the answer snippet as the answer without necessary modification.

Although the two groups' answer correctness rates are on par, finding correct answers in unstructured textual descriptions takes longer time than finding them in the structured representation of activity information in our activity-centric answer snippets. From the task videos, we observe that the control group participants often have to scroll the documents back and forth even though the answer is just in the paragraph they read. Or they often compare several documents to pinpoint and cross-validate the answer information. In contrast, the experimental group participants can view activity information and activity-related code in a more structured way, which helps them find the answers faster, leading to lower task-difficulty ratings and higher answer-confidence ratings. For example, P5 searches both "change fontFamily on TextView" and "change font on TextView" for Task#4. The relevant information for the two queries is actually contained in the same tutorial Fonts in XML. Our activity-centric answer snippets present two different sets of activities centered around two different matched activity frames for the two queries. This helps P5 quickly determine the difference between the two seemingly similar queries and the relevance of the recommended activities to the search task.

Although by no means conclusive due to the small-scale of our study, our study shows that activity-centric knowledge search empowered by the task-oriented knowledge graph is a promising technique to support developer effectively finding task-solving knowledge for programming how-to questions.

VI. RELATED WORKS

Many aspects of API documentation have been well studied, including quality issues [21]–[25], traceability recovery [12], [26]–[30], augmenting API documentation with crowd documentation [31], [32], generating API documentation [33]–[35], and improving the accessibility of API documentation [18]. The recent work by Li et al. [18] is the closet work to ours. Both their work and ours propose knowledge graph based approaches to improve the knowledge accessibility in API documentation. However, Li et al. construct a API caveat knowledge graph with APIs as entities, API caveats as entity attribute, and API dependencies as entity relationships. In contrast, our knowledge graph is task oriented, consisting of task activities, activity attributes (e.g., goal, condition, API, code snippet) and activity relationships (e.g., task hierarchy, temporal dependency). The two types of knowledge graphs are complementary and could be merged in the future.

Recently, several techniques [5], [36], [37] have been developed for paragraph- or sentence-level information retrieval. In terms of information type and granularity, TaskNav by Treude et al. [5] is the closet technique to ours. TaskNav extracts task phrases from textual documentation based on keywords and sentence patterns. In contrast, we train a sentence classifier to detect activity sentences. Furthermore, TaskNav still treats task knowledge as natural language information so it support

task sentence retrieval. In contrast, our work represents task knowledge in a knowledge graph with explicit notion of task activities, activity attributes and activity relationships, and thus we support activity-centric search and knowledge presentation.

Knowledge graphs enable the transition from traditional information retrieval to knowledge search and question answering [11], [38], [39]. In this work, we attempt to achieve this transition for programming task knowledge search by constructing a task-oriented knowledge graph. Open information extraction methods are commonly used to extract knowledge graph entities and relationships from text, for example, DBpedia [40], YAGO [41], and Wikidata [42] from Wikipedia [43]; How-to knowledge graph [44] from WikiHow [45]; and Software Technique Graph [9] from Stack Overflow TagWiki [46]. These existing works inspire the design of our information extraction methods for task-oriented textual tutorials.

Our information extraction methods mix machine learning based and pattern based text analysis. Pattern-based methods are similar to the methods in [5], [9]. Deep learning based activity sentence classification is inspired by the recent advance of deep neural networks in natural language processing [7], [16], [17]. We have also seen the application of deep learning techniques to software text, for example, cross-language question retrieval [47], recommending analogical libraries and APIs [13], [48], classifying self-admitted technical debts [49], and duplicate bug detection [12].

Knowledge graphs, once constructed, can be embedded to obtain vector representations of entities and relationships [50], [51]. These entity and relationship embeddings can be exploited for knowledge reasoning and question answering. For example, Han et al. [52] constructs a software weakness knowledge graph and supports the reasoning of software weakness relationships and consequences based on knowledge graph embedding. This is a promising method we want to apply to task-oriented knowledge graph.

VII. CONCLUSION AND FUTURE WORK

This paper presents a knowledge graph based approach that supports developers "finding activities, not string" for their programming how-to questions. The knowledge graph represents task-solving knowledge in explicit task activities, activity attributes, and activity relationships, extracted from textual programming tutorials using natural language processing techniques. It supports coherent model and matching of how-to questions and task-solving knowledge by well-structured activity frames and addresses the information overload issue by activity-centric search. Our experiments confirm the effectiveness of our knowledge graph construction method and the usefulness of knowledge graph based activity-centric search for solving programming how-to questions. To generalize our results and findings, we will apply our approach to more programming tutorials and release our knowledge graph and search engine for public evaluation. Furthermore, we will combine our knowledge graph with other complementary software-specific knowledge graphs to build a comprehensive software engineering knowledge graph and support advanced knowledge embedding and applications.

REFERENCES

- [1] R. Singh and N. S. Mangat, *Elements of Survey Sampling*. Dordrecht ; Boston: Kluwer Academic Publishers, 1996.
- [2] B. Xu, Z. Xing, X. Xia, D. Lo, and S. Li, "Domain-specific cross-language relevant question retrieval," *Empirical Software Engineering*, vol. 23, no. 2, pp. 1084–1122, 2017.
- [3] Q. Huang, X. Xia, Z. Xing, D. Lo, and X. Wang, "Api method recommendation without worrying about the task-api knowledge gap," in *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, ser. ASE 2018. New York, NY, USA: ACM, 2018, pp. 293–304.
- [4] C. Parnin, C. Treude, and L. Grammel, "Crowd documentation: Exploring the coverage and the dynamics of api discussions on stack overflow," georgia institute of technology," Tech. Rep., 2012.
- [5] C. Treude, M. P. Robillard, and B. Dagenais, "Extracting development tasks to navigate software documentation," *IEEE Trans. on Software Engineering*, vol. 41, no. 6, pp. 565–581, 2015.
- [6] M. Schuster and K. Paliwal, "Bidirectional recurrent neural networks," *Trans. Sig. Proc.*, vol. 45, no. 11, pp. 2673–2681, 1997.
- [7] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: pre-training of deep bidirectional transformers for language understanding," *CoRR*, 2018.
- [8] C. D. Manning, M. Surdeanu, J. Bauer, J. Finkel, S. J. Bethard, and D. McClosky, "The Stanford CoreNLP natural language processing toolkit," in *Association for Computational Linguistics (ACL) System Demonstrations*, 2014, pp. 55–60.
- [9] X. Zhao, Z. Xing, M. A. Kabir, N. Sawada, J. Li, and S.-W. Lin, "HDSKG: Harvesting domain specific knowledge graph from content of webpages," in *2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 2017.
- [10] D. Ye, Z. Xing, C. Y. Foo, J. Li, and N. Kapre, "Learning to extract api mentions from informal natural language discussions," in *2016 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 2016, pp. 389–399.
- [11] D. Lukovnikov, A. Fischer, J. Lehmann, and S. Auer, "Neural network-based question answering over knowledge graphs on word and character level," in *Proceedings of the 26th International Conference on World Wide Web*, ser. WWW '17. Republic and Canton of Geneva, Switzerland: International World Wide Web Conferences Steering Committee, 2017, pp. 1211–1220.
- [12] X. Ye, H. Shen, X. Ma, R. Bunescu, and C. Liu, "From word embeddings to document similarities for improved information retrieval in software engineering," in *Proceedings of the 38th International Conference on Software Engineering*. ACM, 2016, pp. 404–415.
- [13] C. Chen, Z. Xing, Y. Liu, and K. Long Xiong Ong, "Mining likely analogical apis across third-party libraries via large-scale unsupervised api semantics embedding," *IEEE Transactions on Software Engineering*, pp. 1–1, 2019.
- [14] K. Singh, I. Sen, and P. Kumaraguru, "Language identification and named entity recognition in hinglish code mixed tweets," 2018.
- [15] M. A. Hearst, "Support vector machines," *IEEE Intelligent Systems*, vol. 13, no. 4, pp. 18–28, 1998.
- [16] Y. Kim, "Convolutional neural networks for sentence classification," *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*, 2014.
- [17] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," *CoRR*, 2017.
- [18] H. Li, S. Li, J. Sun, Z. Xing, X. Peng, M. Liu, and X. Zhao, "Improving API caveats accessibility by mining API caveats knowledge graph," in *2018 IEEE International Conference on Software Maintenance and Evolution, ICSME 2018, Madrid, Spain, September 23-29, 2018*, 2018, pp. 183–193.
- [19] J. T. Burke, "Utilizing feature location techniques for feature addition and feature enhancement," in *Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering*, ser. ASE '14. New York, NY, USA: ACM, 2014, pp. 879–882.
- [20] N. Alhindawi, N. Dragan, M. L. Collard, and J. I. Maletic, "Improving feature location by enhancing source code with stereotypes," in *2013 IEEE International Conference on Software Maintenance*. IEEE, 2013.
- [21] M. P. Robillard, "What makes apis hard to learn? answers from developers," *IEEE Software*, vol. 26, no. 6, pp. 27–34, 2009.
- [22] G. Uddin and M. P. Robillard, "How api documentation fails," *IEEE Software*, vol. 32, no. 4, pp. 68–75, 2015.
- [23] B. A. Myers and J. Stylos, "Improving api usability," *Communications of the ACM*, vol. 59, no. 6, pp. 62–69, 2016.
- [24] D. Ko, K. Ma, S. Park, S. Kim, D. Kim, and Y. Le Traon, "Api document quality for resolving deprecated apis," in *Proceedings of the 21st Asia-Pacific Software Engineering Conference*, vol. 2. IEEE, 2014, pp. 27–30.
- [25] M. P. Robillard and R. Deline, "A field study of api learning obstacles," *Empirical Software Engineering*, vol. 16, no. 6, pp. 703–732, 2011.
- [26] S. Subramanian, L. Inozemtseva, and R. Holmes, "Live api documentation," in *Proceedings of the 36th International Conference on Software Engineering*. ACM, 2014, pp. 643–652.
- [27] M. P. Robillard and Y. B. Chhetri, "Recommending reference api documentation," *Empirical Software Engineering*, vol. 20, no. 6, pp. 1558–1586, 2015.
- [28] B. Dagenais and M. P. Robillard, "Recovering traceability links between an api and its learning resources," in *Proceedings of the 34th International Conference on Software Engineering*. IEEE, 2012, pp. 47–57.
- [29] P. C. Rigby and M. P. Robillard, "Discovering essential code elements in informal documentation," in *Proceedings of the 35th International Conference on Software Engineering*. IEEE, 2013, pp. 832–841.
- [30] X. Chen and J. Grundy, "Improving automated documentation to code traceability by combining retrieval techniques," in *Proceedings of the 26th International Conference on Automated Software Engineering*. IEEE Computer Society, 2011, pp. 223–232.
- [31] C. Chen and K. Zhang, "Who asked what: Integrating crowdsourced faqs into api documentation," in *Companion Proceedings of the 36th International Conference on Software Engineering*. ACM, 2014, pp. 456–459.
- [32] E. C. Campos, L. B. Souza, and M. d. A. Maia, "Searching crowd knowledge to recommend solutions for api usage tasks," *Journal of Software: Evolution and Process*, vol. 28, no. 10, pp. 863–892, 2016.
- [33] P. W. McBurney and C. McMillan, "Automatic documentation generation via source code summarization of method context," in *Proceedings of the 22nd International Conference on Program Comprehension*. ACM, 2014, pp. 279–290.
- [34] M. L. Collard, M. J. Decker, and J. I. Maletic, "srcml: An infrastructure for the exploration, analysis, and manipulation of source code: A tool demonstration," in *Proceedings of the 29th International Conference on Software Maintenance*. IEEE, 2013, pp. 516–519.
- [35] X. Peng, Y. Zhao, M. Liu, F. Zhang, Y. Liu, X. Wang, and Z. Xing, "Automatic generation of API documentations for open-source projects," in *2018 IEEE Third International Workshop on Dynamic Software Documentation (DySDoc3)*. IEEE, sep 2018.
- [36] B. Xu, Z. Xing, X. Xia, and D. Lo, "Answerbot: Automated generation of answer summary to developers' technical questions," in *Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering*, ser. ASE 2017, Piscataway, NJ, USA, 2017, pp. 706–716.
- [37] H. Jiang, J. Zhang, Z. Ren, and T. Zhang, "An unsupervised approach for discovering relevant tutorial fragments for apis," in *Proceedings of the 39th International Conference on Software Engineering*, ser. ICSE '17, Piscataway, NJ, USA, 2017, pp. 38–48.
- [38] P. Trivedi, G. Maheshwari, M. Dubey, and J. Lehmann, "LC-QuAD: A corpus for complex question answering over knowledge graphs," in *Lecture Notes in Computer Science*. Springer International Publishing, 2017, pp. 210–218.
- [39] Z. Dai, L. Li, and W. Xu, "CFO: conditional focused neural question answering with large-scale knowledge bases," *CoRR*, 2016.
- [40] J. Lehmann, R. Isele, M. Jakob, A. Jentzsch, D. Kontokostas, P. N. Mendes, S. Hellmann, M. Morsey, P. van Kleef, S. Auer, and C. Bizer, "DBpedia - a large-scale, multilingual knowledge base extracted from wikipedia," *Semantic Web Journal*, vol. 6, no. 2, pp. 167–195, 2015.
- [41] F. M. Suchanek, G. Kasneci, and G. Weikum, "Yago: A core of semantic knowledge," in *Proceedings of the 16th International Conference on World Wide Web*, ser. WWW '07. New York, NY, USA: ACM, 2007, pp. 697–706.
- [42] D. Vrandečić and M. Krötzsch, "Wikidata: A free collaborative knowledgebase," *Commun. ACM*, vol. 57, no. 10, pp. 78–85, 2014.
- [43] W. contributors, "Plagiarism — wikipedia, the free encyclopedia," <https://en.wikipedia.org/w/index.php?title=Plagiarism&oldid=5139350>, 2004, online; accessed 22-July-2004. [Online]. Available: <https://en.wikipedia.org/w/index.php?title=Plagiarism&oldid=5139350>
- [44] C. X. Chu, N. Tandon, and G. Weikum, "Distilling task knowledge from how-to communities," in *Proceedings of the 26th International*

- Conference on World Wide Web*, ser. WWW '17. Republic and Canton of Geneva, Switzerland: International World Wide Web Conferences Steering Committee, 2017, pp. 805–814.
- [45] Wikihow, “wikihow - how to do anything,” <https://www.wikihow.com/Main-Page>, 2005. [Online]. Available: <https://www.wikihow.com/Main-Page>
 - [46] S. Overflow, “Stack overflow - where developers learn, share, & build careers,” <https://stackoverflow.com/>, 2008. [Online]. Available: <https://stackoverflow.com/>
 - [47] G. Chen, C. Chen, Z. Xing, and B. Xu, “Learning a dual-language vector space for domain-specific cross-lingual question retrieval,” in *2016 31st IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2016, pp. 744–755.
 - [48] C. Chen, S. Gao, and Z. Xing, “Mining analogical libraries in q&a discussions—incorporating relational and categorical knowledge into word embedding,” in *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, vol. 1. IEEE, 2016, pp. 338–348.
 - [49] Q. Huang, E. Shihab, X. Xia, D. Lo, and S. Li, “Identifying self-admitted technical debt in open source projects using text mining,” *Empirical Software Engineering*, vol. 23, no. 1, pp. 418–451, 2017.
 - [50] A. Bordes, N. Usunier, A. Garcia-Duran, J. Weston, and O. Yakhnenko, “Translating embeddings for modeling multi-relational data,” in *Advances in Neural Information Processing Systems 26*, C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2013, pp. 2787–2795.
 - [51] A. Bordes, S. Chopra, and J. Weston, “Question answering with sub-graph embeddings,” *CoRR*, 2014.
 - [52] Z. Han, X. Li, H. Liu, Z. Xing, and Z. Feng, “DeepWeak: Reasoning common software weaknesses via knowledge graph embedding,” in *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 2018.