2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)

Extracting API Tips from Developer Question and Answer Websites

Shaohua Wang
Corresponding Author, *Department of Informatics*
New Jersey Institute of Technology
Newark NJ, US. davidsw@njit.edu

NhatHai Phan
Department of Informatics
New Jersey Institute of Technology
Newark NJ, US. phan@njit.edu

Yan Wang
School of Information
Central University of Finance and Economics
Beijing, China. dayanking@gmail.com

Yong Zhao
Department of Informatics
New Jersey Institute of Technology
Newark NJ, US. yz746@njit.edu

***Abstract*—The success of question and answer (Q&A) websites attracts massive user-generated content for using and learning APIs, which easily leads to *information overload*: many questions for APIs have a large number of answers containing useful and irrelevant information, and cannot all be consumed by developers. In this work, we develop DeepTip, a novel deep learning-based approach using different Convolutional Neural Network architectures, to extract short practical and useful tips from developer answers. Our extensive empirical experiments prove that DeepTip can extract useful tips from a large corpus of answers to questions with high precision (i.e., avg. 0.854) and coverage (i.e., 0.94), and it outperforms two state-of-the-art baselines by up to 56.7% and 162%, respectively, in terms of Precision. Furthermore, qualitatively, a user study is conducted with real Stack Overflow users and its results confirm that tip extraction is useful and our approach generates high-quality tips.**

***Index Terms*—API, deep learning, CNN, tip extraction, sentence classification**

I. Introduction

Application Programming Interfaces (APIs) are important in modern software development [1]. It is highly desired to use and learn APIs effectively and accurately in software development [2]. Developer Question and Answer (Q&A) websites, such as Stack Overflow[1] (SO), have become essential on-line forums where developers can share their knowledge of using API features, make discussions on the issues about working with APIs, and more importantly learn APIs through other developers' questions and answers [3].

The success of developer Q&A websites, such as SO, has attracted the vast amount of user-generated posts for APIs, including questions, answers, and comments, from developers [4], also creates *information overload* as is the case with other types of user-generated content [5]. A large number of answers and comments accumulated to the questions of popular APIs, with each answer usually containing over a dozen of sentences, make them practically impossible to consume. Furthermore, the growing number of APIs of a programming language and the sheer volume of knowledge about any given API scattered across many distinct posts lead to a significant challenge to quickly gain concise and key insights of APIs [1].

Developer Q&A websites, such as SO, can have moderators and techniques (e.g., a voting-based reward system using reputation, favorites, votes, and badges) to deal with spam, off-topic questions, duplicates and worst-rated posts for ensuring the quality of questions and answers. However, a large amount of user-generated posts can contain some useful information and irrelevant one as well, making the problem of *information overload* even worse [6]. As a result, developers often dig into massive information to pinpoint their needs, and yet, may still miss helpful information.

To help developers navigate through massive information, a lot of research has been made to distill information from developer Q&A websites, especially SO posts, such as summarizing opinions of APIs (e.g., [1]), extracting useful sentences and code examples to improve API documentation (e.g., [7]), identifying malware capabilities (e.g., [8]), mining code snippets and comments (e.g., [9], [10]).

In this paper, we build a Convolutional Neural Network (CNN) based method, namely DeepTip, to extract useful tips for using and learning APIs from a large collection of posts on developer Q&A websites. We focus on Stack Overflow (SO), one of the most active Q&A websites in the software community [11]. We refer to a *tip* as a small, but informative, actionable, and useful piece of information [5] that can help developers understand key insights of answers to a question on Q&A websites. The size of a tip can be varied, ranging from a small phrase having a few words to a paragraph containing several sentences. We argue that in certain cases, developers can be interested in such tips, rather than the entire content of an answer, which often contains lengthy descriptions in dozens of sentences or paragraphs and irrelevant content. The extracted tips can be used to complement the functions of Q&A websites. For example, tips can be highlighted inside the answers of a thread relevant to an API.

[1] $https://stackoverflow.com/$

2574-3864/19/$31.00 ©2019 IEEE
DOI 10.1109/MSR.2019.00058

Through a small pilot user study on SO posts, we find that several sentences within a paragraph can be seen useless individually, but as a whole, they become a *tip*. In this paper, we extract one-sentence and one-paragraph tips for APIs.

To our best knowledge, the closest work to our study is that of Treude et al. [7]. They propose a Supervised Insight Sentence Extractor (*SISE*) using meta features (e.g. user attributes, answer attributes) to extract insightful sentences from SO to augment Java API documentations. Our approach is different from theirs in several ways: (1) DeepTip's goal is to extract useful tips for studying answers to API related questions; (2) SISE is designed for one-sentence tips, not multi-sentence ones; and (3) DeepTip utilizes deep learning techniques and includes more comprehensive features (e.g. word embedding centroid, CNN intermediate feature maps).

Given a corpus of annotated sentences and paragraphs (i.e., each sentence or paragraph is annotated with tip or non-tip) and its related SO posts, DeepTip first learns word embeddings from all of the words in the SO posts and learns the semantics of words in each annotated sentence and paragraph by embedding them into a vector representation using the learned word embeddings. Second, DeepTip trains different multi-layer neural network language models on the top of learned word embeddings. Lastly, given a sentence (S) or a paragraph (P) containing sentences from a post regarding an API, DeepTip classifies an S or a P as a *tip* or not.

In this paper, we extract tips from answers to the questions relevant to 48 frequently used PHP API methods. To evaluate the effectiveness of DeepTip, we process over 1.9 million PHP threads (i.e., a thread has a question and its answers) containing over 3 million posts and link the relevant threads to the 48 PHP API methods. Through empirical studies, we compare DeepTip with two state-of-the-art methods: Supervised Insightful Sentence Extractor (SISE) [7] and Travel Tips Extraction (TTE) [5]. Our results show that DeepTip is effective in tip extraction with high precision (i.e., 0.82-0.857 on paragraphs and 0.81-0.835 on sentences) and high coverage (i.e., 0.71-0.94 on both levels). DeepTip outperforms the SISE and TTE by up to 162% and 56.7% in terms of Precision on tips extraction, respectively. Furthermore, we conduct a comparative user study with 12 SO users to gain feedback on the results of DeepTip and the baselines. The results show that all participants consider our DeepTip is better than the baselines. The main contributions of our work are as follows:

- To our best knowledge, we are the first to propose a deep learning based approach to extract API tips at sentence and paragraph levels from developer Q&A websites.
- We design a parallel one-hot encoding based CNN and a framework to learn embeddings from unlabeled data to improve the performance of CNN on tip extraction.
- We evaluate DeepTip with extensive empirical experiments quantitatively and a comparative user case study qualitatively. Our results show that DeepTip can extract meaningful and useful tips from user-generated answers.
- We are the first to build a large labeled dataset (i.e., 31,046 sentences and 21,895 paragraphs) as a benchmark for tip extraction. Replication Package is online.[2].

Paper Organization. Section II provides the basics of deep learning and CNN. Section III shows a pilot empirical study on analyzing API tips from SO. Section IV presents our deep learning based approach for extracting API tips. Section V and VI present our experimental setup and results. Section VII further discusses our findings and implications. The threats to validity of our work are discussed in Section VIII. Section IX summarizes related work. Finally, we conclude the paper and provide insights for future work in Section X.

II. Background

Our work adopts and augments recent advanced techniques in deep learning and language modeling [12], [13]. We exploit two techniques: word embeddings and CNNs. In this section, we introduce the basics of these two technologies.

Word Embedding. In this paper, DeepTip uses two methods of embedding: one-hot encoding and Word2Vector.

One-hot Encoding (OHE) is a representation of categorical variables as binary vectors, and a popular embedding in language models. Each categorical variable is mapped to integer values. Each integer is represented as a binary vector consisting of all elements 0s but a single element with 1. The OHE can be used to distinguish each word in a vocabulary from every other word in the vocabulary [14]. The OHE makes the representation of words more expressive, but it has the problems of the curse of dimensionality and loss of word order.

Word2Vector (W2V). Several techniques of learning a distributed representation for words were developed [13], [15]–[19]. Among them, W2V [13], [17] is a neural network that takes a text corpus as an input, and generates a set of feature vectors for words in the corpus. It can use two architectures to learn the underlying word representations of a corpus: continuous bag of words (CBOW) and skip-gram. W2V learns high-quality low-dimensional word vectors and can capture syntactic and semantic relationships among words.

Convolutional Neural Network (CNN). The CNN, a feed-forward neural network with convolution, pooling and fully connected layers, has been applied and works extremely well on sentence classification [12], [20]. Different embeddings of a word (e.g., W2V) can be used to construct a document matrix $\mathbf{D}$, where it has a structure $n \times d$ of $\mathbf{D}$ with only one channel (d is the size of word embedding). A convolution operation with a filter can be applied and the filter can convolute a window of words (e.g., 3 or 4) to produce a new feature using the following equation: $c_i = \sigma(\mathbf{W} \cdot \mathbf{h}_\ell(x) + \mathbf{b})$, where c_i is the dot product of $h_\ell(x)$ and a filter weight $\mathbf{W}$. $h_\ell(x)$ is a region matrix for the region x at location ℓ. σ is non-saturating nonlinearity $\sigma(x) = \max(0, x)$ [21]. Then the filter can convolute each possible window of words and produce a feature map: $\mathbf{c} = [c_1, c_2, ..., c_i, ...]$.

[2] $https://github.com/DEEPTIPExtraction/DEEPTip$

The weight $\mathbf{W}$ and biases $\mathbf{b}$ are shared during the training process for the same filter, which enables to learn meaningful features regardless of the window and memorizing the location of useful information when appearing. Then max-pooling operation on $\mathbf{c}$ is applied to take the maximum over a window size of $\mathbf{c}$ [22]. Last, the output volume is reshaped into a vector before sent into a linear classifier, in which softmax function outputs the probability distribution over labels. Dropout and L2 regularization can also be employed on a fully connected layer to prevent overfitting [23].

III. Pilot User Study

We conduct an exploratory pilot user study to investigate what tips look like in answers and learn principles for labeling a larger dataset. We first download Stack Overflow (SO) data in XML published in Sept. 2017 from SO[3]. We extract the questions tagged with "PHP" and their answers. In total, we obtain 3,070,491 posts: 1,915,095 questions and 1,155,396 answers (i.e., 1,915,095 threads). We conduct our exploratory user study in the following steps: First, we create a small sample by randomly choosing 400 PHP threads from 1,915,095 threads based on a confidence level 95% and 5% interval. Table I shows the statistics of the selected 400 threads.

TABLE I: Descriptive statistics for the selected 400 threads

type	#
# of answers	954
# of paragraphs in 954 answers	2,836
# of sentences in 954 answers	4,298

We recruit three graduate students with 1-3 years PHP experience to help label the 2,836 paragraphs as shown in Table I. For simplicity, we process a paragraph into an ordered set of sentences using *NLTK sentence tokenization* tool [24], and use a sentence as a basic unit. The annotators review each set of sentences in all answers of a thread independently and decide a sentence (or a set of consecutive sentences within a paragraph) is a tip or not. A tip (i.e., a small, actionable, and useful piece of information) must be either used to solve a specific question or useful for learning APIs in general.

We apply Cohen's Kappa (CK) coefficient to measure inter-annotator agreement. As we have three annotators, we compute the pair-wise CK coefficient and calculate the average value. We obtain an average CK coefficient of 0.68, indicating as fair agreement [25]. When a disagreement occurs, they discuss it to reach an agreement to mitigate the possibility of errors. If still no consensus is made, the majority rule is applied.

In total, we obtain 124 tips: some have only one sentence and others can have multiple sentences. On average, a tip has 1.452 sentences (i.e., about 22 words). The length of a tip can be from 4 to 59 words and the median length is 21. For simplicity, in this study, we work on two level tips: (1) One-sentence tip: an individual sentence is considered as a tip without considering the surrounding sentences; (2) One-paragraph tip: an individual paragraph must consist of at least two sentences and one-sentence or multi-sentence tips.

[3]https://archive.org/details/stackexchange

IV. DeepTip: Deep Learning for Tip Extraction

We introduce the overview of our DeepTip, then delve into the details of it. Overall, DeepTip has two main steps:

- **Pre-processing.** Given an API method name, we need identify Stack Overflow (SO) threads that are related to the method. We propose a set of regular expression rules to link SO posts with PHP API methods following the approach in [7]. Once the relevant threads are identified, we process them into paragraphs.
- **Classification.** Given a corpus of sentences and paragraphs, we propose four CNN based methods to classify sentences and paragraphs.
  - M1. We implement a CNN architecture as the one in [12] on top of word2vec embeddings to classify tips, named W2V-CNN.
  - M2. We design an one-hot encoding CNN architecture having multiple convolution layers in parallel, named OH-CNN.
  - M3. The third method applies traditional machine learning classifiers on top of various features derived from different sources, such as learned feature maps from W2V-CNN, text and its formatting of the tip candidates, SO community information.
  - M4. We propose a framework to learn features from unlabeled data and integrate them into a supervised CNN for improving the classification performance.

A. Pre-processing

Our pre-processing approach takes questions with answers as an input, and outputs a list of tip candidates (i.e., paragraphs that may have one-sentence or multi-sentence tips.)

We identify relevant SO threads to PHP methods. In our pilot study (Section III), we process the SO data dump and obtain 1,915,095 PHP threads. We replicate the approach that is used for linking SO threads to Java types in [7]. More specifically, given the corpus of the above threads and a PHP method name, we interpolate the keywords from the method name and build regular expressions containing the keywords, using the approach in [7].

Our built regular expressions for PHP methods are shown in Table II. The regular expressions are built for two parts: the title and body of a post. We use both parts in our linking process. We process the code within html tags of *<pre><code>...</code></pre>* into textual tokens for linking. For a PHP method, we reject all *PHP* questions that cannot match at least one of our regular expressions containing the keywords built for the method. After the linking process, we split the answers of an identified thread into paragraphs using html tags, such as * ... * or *
...
* or *
...
* in SO data.

B. Classification Methods

Once a list of tip candidates are generated in the previous step, DeepTip classifies them to be tips or non-tips. Here, we present four classification methods.

TABLE II: Regular repressions for filtering SO questions

Part	Regular expression (KW = Key Words for a method name)
title	(?i).*\b%KW\b.* (*fully-qualified method, case-insensitive*)
title	(?i).*\b(a |an)%KW\b.* (*non-qualified method prefixed with "a" and "an", case-insensitive*)
body	*(^|[a-z]+|[\.!?\(<\=]\s{0,}|\={0,1}\s{0,}\$[\w\d_]*\-\>) %KW(\s{0,}[\(\[][\w\s$,\'\"\(\)]*[\)\]]\s{1,}\|[>\)\.,!?$]+|[a-z]+).* (*non-qualified method surrounded by lower case words or punctuation signs*)
body	.*<code>.*\b%KW\b.*</code>.*(*non qualified method in code*)
body	.*<a.*href.*%KW\.php.*>.*.* (*PHP manual website host*)

[Method 1.] DeepTip-W2V.

We replicate the word2vec (W2V) CNN architecture studied in [12] to classify tip candidates, namely DeepTip-W2V. More specifically, DeepTip-W2V consists of one convlutional layer with different filter sizes, one max pooling layer, and a fully connected output layer. We use W2V to learn vectors for each word [13], [17]. We remove punctuations and keep stopping words before feeding text into W2V.

[Method 2.] DeepTip-OH.

Inspired by [20], we develop a new CNN architecture on the top of one-hot encodings of words to classify tip candidates. Compared with the simple CNN architecture in the section of background II, we create a parallel One-hot CNN (OH-CNN), namely DeepTip-OH, taking the One-hot encoding of words as input and having three convolutional layers in parallel as shown in Figure 1. For example, the three convolutional layers are the *h1, h2, and h3* in the convolution of Figure 1. Multiple convolutional layers embed different text regions to complement each other to improve performance.

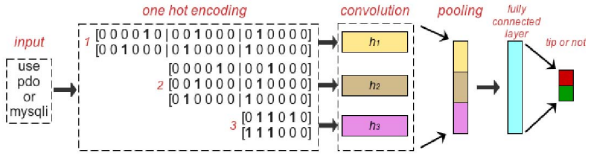

Fig. 1: The architecture of DeepTip-OH.

More specifically, DeepTip-OH directly converts one hot vector of words or a n-gram into feature maps in the convolutional layers [20]. Given a tip candidate $T = (w_1, w_2, ..., w_n)$ with a vocabulary V, we treat each word in T as a pixel in images, and deal with T as if it is an image of size of $|T| \times 1$ with $|V|$ channels. We represent text regions by concatenating continuous words, and then each region vector has a size of $p|V|$ where p is the fixed region size. The size of each convolutional filter should be $p \times 1 \times |V|$. For example, given a $V = \{mysqli, or, pdo, sql, use, your\}$ and a tip candidate "use pdo or mysqli", we have a tip vector:

$$x = [0\ 0\ 0\ 1\ 0 \mid 0\ 0\ 1\ 0\ 0\ 0 \mid 0\ 1\ 0\ 0\ 0\ 0 \mid 1\ 0\ 0\ 0\ 0\ 0]^T \quad (1)$$

The *one hot encoding* part of Figure 1 shows the examples of vectors: if $p = 3$ and stride is 1, we have two regions "use pdo or" and "pdo or mysqli", the corresponding vectors are labeled with 1. Likewise for $p = 2$ and stride is 1, three regions: "use pdo", "pdo or", and "or mysqpli" are attained, the corresponding vectors are labeled with 2. The convolutional layer learns to embed different text regions into a low-dimensional vector space. Such region embeddings, namely *seq-CNN* [20], can preserve the sequence of words.

A potential problem of *seq-CNN* is that a region vector can be high-dimensional if $|V|$ and p are both very large, which not only makes it hard to learn parameters, but lead to overfitting as well. A workaround is that we can perform bag of words conversion to make region vectors. Then the size of a region vector is always $|V|$ regardless of p. For example, in the *one hot encoding* part of Figure 1, the last two region vectors with $p = 3$ are noted by number 3 . Further, bag-of-word can be extended to bag-of-n-grams. The convolutional layer learns region embedding from one-hot vector with n-gram vocabulary. A bag-of-n-grams scheme has the simplicity of the bag-of-words model, but allows the preservation of more word locality information. This is *bow-convolution* [20].

The sizes of tip candidates are naturally varied. Thus, with the same stride, convolutional layers output varied sizes of feature maps. Therefore, our DeepTip-OH has two seq-convolution and one bow-convolution layers in parallel. In *OH-CNN*, multiple pooling units can be applied to one entire tip candidate. Also, we can use different pooling techniques (e.g., max and average poolings). The top layer is a fully connected layer.

The differences between the DeepTip-W2V and DeepTip-OH are: (1) The W2V-CNN employs one convolution layer with different filter sizes; The DeepTip-OH have three convolution layers with different text region sizes in parallel. (2) The W2V-CNN is built on the top of an additional word vector conversion layer (i.e., word2vec);(3) The W2V-CNN pads tip candidates to have a same vector length; and (4) The W2V-CNN applies the max-pooling on entire test with one unit.

[Method 3.] DeepTip-Features (F).

We extract various types of features from tip candidates and their community information (e.g., user reputation and up-votes), then apply a machine learning classifier on the top of extracted features to classify a tip candidate.

Extracting features for DeepTip-f. Table III shows all of the features used by our DeepTip-F. The Meta-Set, TGrams, and POSGrams features can be easily and straightforwardly extracted from the SO data. Therefore, due to the page limit, we focus on introducing the following features:

W2V-CNN and OH-CNN features. W2V-CNN features include (1) the intermediate feature maps, the concatenated vectors of different filter sizes fed to a fully connected layer; and (2) the final output of running DeepTip-W2V in the first method. Our OH-CNN features are the final output of running DeepTip-OH.

Two W2V features. UniW2V and IDFW2V are calculated using Equation (2).

$$\overrightarrow{\boldsymbol{UniW2V}} = \frac{\sum_{i=0}^{\ell} \vec{\mathbf{w}}_i}{\ell} \quad \overrightarrow{\boldsymbol{IDFW2V}} = \frac{\sum_{i=0}^{\ell} \vec{\mathbf{w}}_i \times IDF(\mathbf{w}_i)}{\sum_{i=0}^{\ell} IDF(\mathbf{w}_i)} \quad (2)$$

where ℓ is length of tip candidates, $\mathbf{w}_i$ and $\vec{\mathbf{w}}_i$ are ith word in tip candidate and its corresponding vector.

TABLE III: Features used in DeepTip-F

Feature sets	Feature type	Features
Meta-Set	Question	Score, User reputation, Views, Age, Favorites
	Answer	Score, Time difference to question, Age, Size
	Grammar	Number of tokens in tip candidate, Number of nouns in tip candidate, Whether tip candidate starts with noun, Number of characters that are code in tip candidate, Number of occurrences of the verb be in tip candidate
	Similarity	Cosine similarity between a tip candidate and its corresponding question
Language Model	TGrams	Number of occurrences of part-of-speech Unigram, Bigram, and Trigram, we consider Unigram, Bigram, and Trigram occurring at least 5 times in the dataset
	W2V	UniW2V: Uniformly weighted word vectors in each tip candidate, see equation 2
		IDFW2V: Inverse document frequency weighted word vectors in each tip candidate, see equation 2
	POSGrams	Number of occurrences of part-of-speech (POS) Unigrams, Bigrams, and Trigrams in tip candidates. We discard 1-3 POS grams appearing less than 5 times
	Template	IDs of matched templates, Number of templates that a tip candidate matches, Number of tokens in the longest template the tip match
W2V-CNN		Intermediate feature maps, Final outputs
OH-CNN		Final outputs

Template features. Stack Overflow (SO) allows questioners to select one of the posted answers as an accepted answer. To some extent, the accepted answers are the most helpful or suitable answers [26]–[28]. We extract template features from the SO accepted answers using the approach in [5]. We seek for n-grams that repeatedly occur in a number of paragraphs from the accepted answers. Such n-grams are templates.

More specifically, n-grams with 4≤ n≤ 6 are considered in our work, as small values of n can generate too generic patterns whereas large values of n generate rare patterns. To have some level of generalization, we allow one word of a given n-gram to be a *wildcard* (i.e. it may represent any single word) and the *wildcard* can be at any position in a n-gram. For example, in a 4-gram "you need to *", * is the *wildcard* and can stand for 'sanitize', 'indicate', 'use', etc. For each 4≤ n≤ 6, we extract a list of 1-wildcard n-grams from the SO accepted answers. We require a n-gram's wildcard to capture at least two different words across its occurrences. We rank our 1-wildcard n-grams based on the number of occurrences in the accepted answers from high to low. We choose the n-grams with at least N occurrences as *templates*. In this paper, for the simplicity, we set N to be the average number of occurrences (i.e., all the occurrences from all n-grams over the number of n-grams) in the accepted answers. Once we obtain templates, we extract the template-based features as shown in Table III.

[Method 4.] DeepTip-Semi.

The unlabeled data can contain valuable information. We propose DeepTip-SEMI, adopting a state-of-the-art framework proposed by Johnson et al. [29], to learn embeddings of small text regions from unlabeled data and integrate them into a supervised CNN. The learning of textual embeddings is based on the idea of two-view semi-supervised learning [30], [31] used to predict the context from each region of size p continuous words in a convolutional layer. The two-view embedding has the following property [32]:

Definition 1. *A function f_1 is a two-view embedding of χ_1 w.r.t. χ_2 if there exists a function $g1$ such that $P(X_2|X_1) = g_1(f_1(X_1), X_2)$ for any $(X_1, X_2) \in \chi_1 \times \chi_2$.*

For example, in a sentence "your code is prone to sql injection", the clues for predicting the label of "prone to sql" (X_1) are itself and its contexts *is prone to* and *to sql injection* (X_2). A neural network μ is trained to approximate $P(X_2|X_1)$ by $g_1(f_1(X_1), X_2)$ as in Definition 1. f_1 is a convolution layer and g_1 is a fully connected layer.

$$\mu_\ell(x) = \sigma^\mu(\mathbf{W}^\mu \cdot \mathbf{h}^\mu_\ell(x) + \mathbf{b}^\mu) \tag{3}$$

where $\mathbf{h}^\mu_\ell(x)$ can be sequences of words, bag of words, or bag of n-grams. A convolution layer can be defined in Equation 4.

$$c_i = \sigma(\mathbf{W} \cdot \mathbf{h}_\ell(x) + \mathbf{b}) \tag{4}$$

where c_i is the dot product of $h_\ell(x)$ and filter weight $\mathbf{W}$. $h_\ell(x)$ is a region matrix for the region x at location ℓ. σ is non-saturating nonlinearity $\sigma(x) = \max(0, x)$ [21].

In our DeepTip-SEMI, for the supervised part, we produce additional input to Equation 4 and the new convolution layer is rewritten to the following equation:

$$c_i = \sigma(\mathbf{W} \cdot \mathbf{h}_\ell(x) + \mathbf{V} \cdot \mu_\ell(x) + \mathbf{b}) \tag{5}$$

$\mathbf{W}$, $\mathbf{V}$, and $\mathbf{b}$ are trained based on the labeled data.

We can have two ways to train a neural network $\mu_\ell(x)$: unsupervised target and partially-supervised target [29]. For the unsupervised target, it is straightforward as we describe in the above example. We predict context X_2 from X_1 directly only using the text (i.e., text data with labels or just use unlabeled data). One disadvantage is that some irrelevant words can be introduced into region embeddings. To remove the effect of syntactic relations (e.g. "is" is a functional word unrelated to our task), we remove stop words in the context. Another way is to train a CNN with labeled data and apply it to unlabeled data. We remove predictions from the trained CNN and only use the internal output of the convolutional layer. This strategy helps create vectors representing labels (i.e. concepts) very well, but some concepts might only occur in unlabeled data.

In this paper, we employ both methods as they can complement to each other. Given a trained CNN model, when training unsupervised target, we use bag of words and bag of 3-gram to learn the region embeddings. Our additional input has three parts as follows:

$$c_i = \sigma(\mathbf{W} \cdot \mathbf{h}_\ell(x) + \mathbf{V_1} \cdot \mu^{ut}_\ell(x) + \mathbf{V_2} \cdot \mu^{ut3}_\ell(x) + \mathbf{V_3} \cdot \mu^{ps}_\ell(x) + \mathbf{b}) \tag{6}$$

where $\mu^{ut}_\ell(x)$, $\mu^{ut3}_\ell(x)$, and $\mu^{ps}_\ell(x)$ are region embedding models by unsupervised target of bag of words and bag of 3-gram, and partially-supervised target. Weights of $\mathbf{V_1}$, $\mathbf{V_2}$, and $\mathbf{V_3}$ are learned during the training for each unsupervised region embedding.

V. Experimental Setup

We conduct several experiments to evaluate our DeepTip on a server with 16 CPUs with 1200.390 MHz, one TITAN Xp GPU, and 128G RAM. Here, we describe our experimental setup for these experiments.

Data Processing and Golden Dataset. In this paper, we focus on 48 PHP methods (shown in Table IV) as they are mentioned in the posts used in our pilot user case study (Section III).

TABLE IV: 48 commonly used PHP methods

categories	method
global variables (13)	isset, unset, _post, _get, _cookie, _session, _server, _request, filter_var, setcookie, session_start, session_unset, session_destroy
string (7)	explode, str_replace, strpos, strtolower, strtoupper, is_string, implode
array (19)	is_array, in_array, array_merge, array_keys, array_key_exists, key_exists, array_shift, array_push, array_pop, array_values, array_map, array_unique, array_slice, array_diff, array_search, array_reverse, preg_match, array_unshift, array_multisort
database (3)	pdo, mysqli, mysqli_connect
others (6)	htmlentities, htmlspecialchars, strip_tags, addslashes, print_r, var_dump

In our pilot exploratory user study (Section III), we collect 3,070,491 PHP posts. We link the SO posts with our 48 studied PHP API methods using the linking approach in Section IV-A. We obtain 363,834 relevant threads having 592,488 answers; 192,971 out of 363,834 questions have an accepted answer.

Golden Dataset. To evaluate our approaches, we recruit three annotators different from the ones in our pilot user study (Section (III)) to build a golden dataset containing sentences and paragraphs that are labeled with 1s (i.e., tips) or 0s (i.e., non-tips). The three annotators are: the second author of this paper, one software developer with 3 years PHP experience, and a graduate student with 1 year PHP experience. We randomly select 3,000 threads from the above obtained 363,834 threads. Due to limited resources, we select 3000 threads representing the distribution of the 363,834 threads at the 99%$\pm$2.35%, but better than the commonly used sample size at the 95%$\pm$5% (i.e., only requires 384 threads). For each thread, we process its answers into paragraphs using html tags in the post. After processing, if a paragraph only contains one sentence, we consider it as a sentence not a paragraph. We obtain 21,895 unique paragraph-level tip candidates as a dataset *TipPara*. We further process paragraphs in *TipPara* into sentences using *NLTK sentence tokenization* tool [24] and gain 31,046 unique sentence-level tip candidates, which is dataset *TipSent*. Using the learned knowledge of tips in the pilot user study III, each annotator reviews every paragraph and sentence in *TipPara* and *TipSent*, respectively.

Given a sentence or a paragraph, and its corresponding PHP API method and the corresponding question post, three annotators independently and separately decide whether the sentence or paragraph is a tip or not for learning the method or answering the corresponding question. We use Cohen's Kappa (CK) coefficient to measure inter-annotator agreement. We computer the pair-wise CK coefficient and calculate the average CK value for sentences and paragraphs. We obtain an average CK of 0.81 for sentences and 0.85 for paragraphs, indicating as good agreement [25]. When a disagreement occurs, the majority rule is applied. Table V shows the statistics of our annotated tips.

TABLE V: Descriptive Statistics for Tip candidates

dataset	total #	# of tips	# of non-tips
TipPara	21,895	1,156	20,739
TipSent	31,046	1,342	29,704

Evaluation Metrics. We use five measures to evaluate the performance of our DeepTip: AUC [33], precision (P), recall (R), F-measure (F), and Coverage (Cov).

$$AUC = \frac{1}{mn} \times \sum_{i=1}^{m} \times \sum_{j=1}^{n} \mathbf{1}_{p_i > p_j} \quad (7)$$

*where m and n are the numbers of true positive and negative samples; p_i and p_j are the probabilities assigned by a classifier; **1** is the indicator function: it outputs 1 if the condition (here $p_i > p_j$) is satisfied*

$$P = \frac{TP}{TP+FP},\ R = \frac{TP}{TP+FN},\ F = \frac{2 \times P \times R}{P+R} \quad (8)$$

Where TP = True Positives; FP = False Positives; FN = False Negatives; TN = True Negatives; P = Precision; R = Recall.

$$Cov = \frac{Number\ of\ API\ methods\ having\ at\ least\ one\ tip}{Total\ Number\ of\ API\ mehtods} \quad (9)$$

Our goal is to extract high-quality tips that can be consumed by developers in a short time, not to identify all possible tips. In this paper, we aim at high precision, i.e., high likelihood of a tip to be useful, and consider that precision is more important than recall, as false positives will drive people away. As the extracted tips can be used for learning API methods, we also use coverage to measure how good an approach is across a set of API methods. The coverage is defined as the proportion of API methods, for which there is at least one tip is extracted.

Comparison Baselines. We compare our DeepTip approaches with the following two baselines:

1) SISE [7] is developed specifically for extracting insight sentences from Stack Overflow to augment API documentation. SISE builds classification features built upon the sentences themselves, the formatting of the sentences, the questions and answers of the sentences, the authors of the sentences, as well as part-of-speech features and the similarity of a sentence to the corresponding API documentation. Then, SISE tests five classifiers: k-nearest neighbours (KNN) [34], decision tree (DT) [35], Naive Bayes (NB) [32], random forest (RF) [36], and support vector machine (SVM) [37] using the extracted features to identify the best-performing classifier on predicting whether a sentence is useful.

2) TTE [5] mines travel tips from user-generated reviews having several sentences on TripAdvisor. Guy et al. [5] designed the classification features based on four categories of information: the tip's text, the matched templates (i.e., frequently occurred n-grams in reviews), the originating review, and the reviewer. TTE tests three classifiers: Logistic Regression (LR) [38], SVM [37] and Adaptive Regularization of Weights (AROW) [39] to identify the best performing

classifier on predicting whether a sentence is useful. Although the TTE is not designed for StackOverflow, the method of extracting tips in the TTE is a comparable baseline.

VI. Experimental Results

We formulate our experiments to answer three research questions. For each research question, we present our motivation, analysis approach and a discussion of our findings.

RQ1. Can DeepTip extract useful API tips?

Motivation. It is important to evaluate the effectiveness of our proposed approaches through a set of empirical studies.

Approach. We first compare our approaches with the two baselines: SISE and TTE.

Building classification features for SISE. In this paper, we build all features from [7] except the following features: Since SISE try to mine insightful sentences from SO for improving Java API documentation, they calculate the similarity between a sentence and sentences in API documentation, also build features on the number of occurrences and positions of an API appearing in the Java Documentation. However, we aim to extract tips useful for learning a specific question or an API in general. We calculate the similarity between sentences in questions and tip candidates based on *Term Frequency and Inverse Document Frequency* (TF-IDF) [40] using cosine similarity. Meta data features regarding answer, question (e.g. score, views) can be gained directly from PHP posts and we use NLTK [24] to generate part of tags for identifying nouns and "be" in tip candidates for the grammar features.

Building classification features for TTE. Although TTE [5] is built upon the TripAdvisor that is different from the Stack Overflow, most of the classification features from TTE can be transformed into the counterparts on the Stack Overflow. However, some features conceptually loose the classification power, such as one reviewer feature, namely the ratio between the number of "helpful" votes and the total number of reviews, becomes meaningless after the transformation, because only one answer on the Stack Overflow can be accepted (i.e., can be viewed as an official "helpful"). Therefore, we implement most of the classification features from TTE into the features based on the Stack Overflow. The features are all the features in Language Model, some reviewer and review features in Meta-Set as shown in Table III.

Tuning classifiers for SISE and TTE. Treude et al. [7] use the WEKA [41] default settings to train the above mentioned five classifiers and conduct attribute selection [42] to improve classifier performance. Guy et al. [5] use 10-fold cross-validation to tune the hyper-parameters and evaluate the above mentioned three classifiers. In this paper, we run eight classifiers (i.e., 5 classifiers in SISE + 3 classifiers in TTE) for each baseline. We use Auto-Weka [43], a system designed to automatically perform attribute selection and tuning hyperparameters, to maximize the classifier performance, except for KNN and AROW, as they are not supported by Auto-Weka. We use WEKA and LIBSOL [44] (i.e., open-source library for online learning) to train and run KNN and AROW, respectively.

TABLE VI: Hyper-parameter settings for each algorithm

Algorithm	Hyper-parameter setting
Word2Vec	**Tuned** {window size = 5, vector size = 200, epoch = 100, skipgram model, hierarchical softmax}. Using default settings for other parameters in gensim.
DeepTip-OH-CNN	**Tuned**{epoch = 165, dropout = 0.5, ReLU, one-unit-max-pooling, seq-CNN node number = 1000, bow-CNN node number = 20, learning rate = 0.25, batch_size = 32 region size = 3,4}. Using default settings for other parameters as recommended in the original paper [20]
DeepTip-W2V-CNN	**Tuned** {kernel size = 3,4,5, epoch = 50, l2 regularization = 3.0, filter number = 128, batch size = 128, text length = 100}. Word embeddings as input and kept same during the training, using default settings for other parameters as recommended in the original paper [12].

Tuning our DeepTip approaches. We set each approach on sentence and paragraph levels to have the same values for hyper-parameters. We use word2vec [13], [17] in gensim to learn word embeddings. Gensim [45] is a robust open-source vector space modeling and topic modeling toolkit implemented in Python. We tune the models with several hyper-parameters, such as epoch size, filter numbers (i.e., 32, 64,128), learning rates. Due to the page limit, we only report the best settings in Table VI. Also Table VI shows which hyper-parameters were tuned for each approach.

TABLE VII: The results of our DeepTip approaches and the baselines. We run 8 classifiers for each baseline (i.e., SISE and TTE) and 5 classifiers for DeepTip-F. Due to the page limit, we only report the results of a classifier performing the best in terms of at least one metric for each baseline.

Level	Classifier	AUC	P	R	F	C
Sentence	**Baselines**					
	SISE-RF	**0.623**	**0.360**	0.109	0.167	0.330
	SISE-NB	0.574	0.212	**0.447**	**0.287**	**0.460**
	TTE-LR	**0.720**	**0.65**	0.38	**0.480**	0.600
	TTE-AROW	0.580	0.525	**0.440**	0.479	**0.625**
	Ours					
	DTF-LR	**0.876**	0.738	0.347	0.472	**0.710**
	DTF-NB	0.874	0.497	**0.581**	0.535	0.580
	DTF-SVM	0.606	**0.819**	0.268	0.404	0.540
	DeepTip-W2V	0.835	0.730	0.410	0.525	0.81
	DeepTip-OH	**0.872**	**0.851**	**0.49**	**0.622**	**0.85**

Level	Classifier	AUC	P	R	$F_{0.5}$	C
Paragraph	**Baselines**					
	SISE-DT	0.595	0.267	**0.280**	**0.273**	**0.375**
	SISE-RF	**0.610**	**0.342**	0.12	0.178	0.33
	TTE-LR	**0.705**	**0.582**	**0.216**	**0.315**	**0.5**
	Ours					
	DTF-LR	**0.850**	0.750	0.418	0.537	**0.77**
	DTF-NB	0.843	0.668	**0.459**	**0.544**	0.625
	DTF-SVM	0.639	**0.827**	0.326	0.468	0.6
	DeepTip-W2V	0.842	0.755	**0.426**	0.545	0.83
	DeepTip-OH	**0.865**	**0.857**	0.42	**0.564**	**0.875**

P = Precision, **R** = Recall, **F** = F-score, **C** = Coverage
LR = Logistic Regression, **DT** = Decision Tree, **DTF** = DeepTip-F
RF =Random Forest, **NB** = Naive Bayes
AROW = Adaptive Regularization of Weights

Tuning classifiers for DeepTip-F. We build DeepTip-F models using the above five algorithms: LR, DT, NB, RF, and SVM on top of all features proposed in Table III. We use Auto-Weka to maximize the performance of classifiers. To build template

TABLE VIII: The impact of types of features on DeepTip-F using **Logistic Regression** on *sentence- and paragraph- level tips*. P = Precision, R = Recall, F = F-score, C = Coverage, — = exclude

Feature Sets	Sentence Level					Paragraph Level				
	AUC	**P**	**R**	**F**	**C**	**AUC**	**P**	**R**	**F**	**C**
all features	0.876	0.738	0.347	0.472	0.71	0.850	0.750	0.418	0.537	0.77
— W2V-CNN	**0.605**	**0.500**	**0.014**	**0.027**	**0.188**	**0.607**	0.792	**0.038**	**0.072**	**0.21**
— Meta-Set	0.857	0.612	0.376	0.466	0.67	0.847	**0.674**	0.464	0.55	0.75
— TGrams	0.875	0.730	0.345	0.469	0.69	0.851	0.746	0.409	0.528	0.77
— POSGrams	0.876	0.730	0.343	0.467	0.69	0.850	0.733	0.414	0.529	0.77
— template	0.875	0.723	0.357	0.478	0.67	0.850	0.750	0.421	0.539	0.71
— OH-CNN	0.875	0.730	0.356	0.479	0.71	0.851	0.742	0.421	0.537	0.77
— Uni & IDFW2V	0.877	0.717	0.362	0.481	0.67	0.851	0.739	0.426	0.54	0.75
— UniW2V	0.876	0.721	0.358	0.478	0.67	0.851	0.745	0.425	0.541	0.77
— IDFW2V	0.876	0.719	0.353	0.474	0.69	0.85	0.753	0.416	0.536	0.77

features for DeepTip-F, we choose the n-grams with at least 800 occurrences (i.e., the avg. number of occurrences in our dataset) as templates. To investigate the impact of a type of features, we exclude that type of features when building and testing a model.

We run all of the approaches on our datasets *TipPara* and *TipSent* to answer RQ1. For all experiments, we use the same data splitting scheme: 90% for training and 10% for testing, and a 10-fold cross validation. The splitting scheme is common in deep learning based approaches (e.g., [46]). We conduct 10 repeats of our cross validation.

Results of RQ1. We organize our empirical results and discussions as follows:

Our DeepTip approaches outperform the baselines, and DeepTip-OH performs the best. We obtain the following observations from Table VII:

[O1.] Among the SISE baselines, the SISE-RF works the best in terms of Precision. However, the SISE-NB and SISE-DT perform the best on sentences and paragraphs in terms of F-score (F), respectively. Among the TTE baselines, the TTE-LR outperforms others at sentence- and paragraph- levels in terms of F. *Among all the baselines, the TTE-LR is the best.*

[O2.] Our DeepTip-OH can outperform any other models, including other DeepTip models, with a high precision (e.g., 0.857) and coverage (e.g., 0.875). DeepTip-OH improves the best baseline TTE-LR by 31%, 30%, and 42% at sentence-level; and by 47%, 0.79%, and 75% at paragraph-level, in terms of Precision, F and Coverage. Comparing with SISE and TTE baselines, the extracted tips of DeepTip-OH are statistically significantly different from those of the other two baselines (Pearson's chi square, $p<0.005$).

On average, it costs avg. $\sim$2 hours to train our DeepTip-OH model and avg. $\sim$20 mins to train the TTE-LR. The TTE' W2V features require the training of W2V on all the words in the relevant PHP posts (over 900k), but DeepTip-OH does not have the additional conversion layer on top of W2V. DeepTip-OH can be trained off-line when it is applied in the real-world usage scenario.

DEEPTip-F models with features learned from multiple aspects, such as community info (vote counts and user info) do not get consistent and superior performance compared with only CNNs. Among the DeepTip-F models, the Logistic Regression achieves the best results on both levels in terms of Precision, F, and Coverage.

The analysis of features' impact on DeepTip-F.

We train DeepTip-F using a Logistic Regression classifier using different sets of features. Our results in Table VIII show that **semantics features learned by CNN and SO community meta data (e.g., user's reputation and upvotes) have a great impact our DeepTip-F**.

On sentence-level tips, every type of features have an impact on the overall performance of DeepTip-F, especially W2V-CNN and Meta-Set features. More specifically, removing the learned features from W2V-CNN can decrease the overall performance on sentence-level tips in every measurement metric. On paragraph-level tips, removing W2V-CNN decreases the performance in every measurement metric except for Precision, while Meta-Set reduces the precision by 10.1%.

RQ2. Can learning unlabeled data improve DeepTip?

Motivation. Manually labeling data is a labor-intense job, we can only label a small portion of our dataset. In this RQ, we investigate the performance of DeepTip-SEMI to see whether unlabeled data can help improve the classifier performance.

Approach. We choose DeepTip-OH, our best-performing model, as the trained CNN model for DeepTip-SEMI. As discussed in Section V, we obtain 363,834 threads in total and 3000 of them are manually labeled. Thus, we use the rest 360,834 threads to learn small text regions for DeepTip-SEMI. We tune DeepTip-SEMI with different hyper-parameters. The best parameter setting is: region size = 4, epoch = 35, node = 1000, unsupervised learning rate = 0.5, supervised learning rate = 0.25. DeepTip-SEMI at sentence and paragraph levels share same hyper-parameters.

TABLE IX: Performance of the semi-supervised DeepTip-SEMI on sentences and paragraphs

level	**method**	**AUC**	**P**	**R**	**F**	**C**
sentence	DeepTip-OH	0.872	0.851	0.49	0.622	0.85
	DeepTip-SEMI	0.900 **+3.2%**	0.825 **-3.1%**	0.691 **+41%**	0.752 **+20.9%**	0.94 **+10.6%**
paragraph	DeepTip-OH	0.865	0.857	0.42	0.564	0.875
	DeepTip-SEMI	0.895 **+3.5%**	0.82 **-4.3%**	0.580 **+38.1%**	0.647 **+14.7%**	0.94 **7.4%**

Results of RQ2: Table IX shows that our DeepTip-SEMI can leverage unlabeled data to improve the overall performance. The DeepTip-SEMI can improve DeepTip-OH in terms of every metric, but Precision. We conclude that **incorporating un-labeled data can help improve the overall performance of DEEPTip, but brings in noise to decrease the precision**. In terms of F, *DeepTip-SEMI* improves *DeepTip-OH* by 20.7% and 14.7% on sentences and paragraphs, respectively. In terms of Recall, *DeepTip-SEMI* improves *DeepTip-OH* by 41% and 38.1% on sentences and paragraphs, respectively

RQ3. Do developers find the mined tips useful?

Motivation. We test DeepTip qualitatively by conducting a comparative user study. We set out to collect feedback from real developers on the effectiveness of tip extraction from DeepTip and the state-of-the art baselines.

Approach. We send out the user study hiring information on SO Slack, and it was on Slack for 2 weeks. We also sent out our hiring information to the school emailing lists. Initially, 12 SO users from Slack Community of PHP [47] or web development and school emailing lists agreed to participate our study. We create a website for labeling. If one replies, we e-mail the participant with the credentials. A participant can log in and out (even quit the labeling process) anytime during the labeling process and there were no time constraints. To minimize the bias, we did not explain the research goal to participants. Table X shows the basic information of the 12 participants: 8 of them are from SO Slack and 4 of them are from school emailing lists.

TABLE X: Participants in the comparative study

Occupation	PHP Experience	#
Software Engineer	1-3 years	3
Student	1-3 years	6
Student	less than 1 years	3

TABLE XI: Comparative study results. LR=Logistic Regression, NB=Naive Bayes.

level	Sentence # of True Tips vs Total	Paragraph # of True Tips vs Total
DeepTip	62 vs 73 (85%)	68 vs 79 (86%)
TTE-LR	40 vs 71 (56%)	42 vs 70 (60%)
SISE-NB	18 vs 56 (32%)	16 vs 52 (31%)

We randomly select 200 threads from our unlabeled dataset. We run the trained DeepTip-SEMI, TTE-LR, and SISE-RF in RQ1 and RQ2 on the selected threads to generate tips. Then the participants review the generated tips to identify **True Tips**. More specifically, for each thread, we show participants the question, the relevant API method name, and the tips generated from the above three methods. All the unique one-sentence and one-paragraph extracted tips of a thread are shown together to the participants. The participants have no knowledge of the tips belonging to which approach. For each extracted tip, a participant will choose (a) *Tip* or (b) *Not Tip*. After the review, we ask participants to answer two general questions: What is your occupation? and How many years of experience in web application development or PHP do you have? Each extracted tip is reviewed by three different participants. We use the majority-win rule.

Results of RQ3: Table XI shows that **DeepTip-SEMI outperforms the TTE and SISE**. Our DeepTip-SEMI identifies 62 true tips out of the extracted 73 sentence tips (i.e., 85%), and 68 true tips out of the extracted 79 paragraph tips (ie.e, 86%). However, only 56% of the extracted 71 sentence tips, and 60% of the extracted 70 paragraph tips, from the TTE, are reviewed as true tips. Even worse, only 32% of the extracted 56 sentence tips, and 31% of the extracted 52 paragraph tips, from the SISE, are true tips. The review results on our approach and the baselines are consistent with our empirical results in **RQ1** and **RQ2**.

We use Cohen's Kappa (CK) coefficient to measure inter-annotator agreement. We compute the pair-wise CK coefficient and calculate the average CK value for sentences and paragraphs. We obtain an average CK of 0.82 on sentences and 0.84 on paragraphs for DeepTip, and an average CK of 0.80 on sentences and 0.81 on paragraphs for TTE, indicating as good agreement [25]. The obtained average CKs also confirm the high quality of our labeled golden dataset in Section V (i.e., having CKs over 0.8).

VII. Discussion and Implication

A big challenge for tip extraction is to learn semantic discriminative differences between tips and non-tips. Existing methods using manually crafted features derived from the community meta-data and post textual information normally lack a deep understanding of high-level semantics of tips, as the manually crafted features cannot capture complex and deep latent linguistic patterns.

A significant difference between DeepTip-OH and TTE (or SISE) is that DeepTip-OH embeds words into a high-dimensional space using one-hot encoding. DeepTip-OH directly applies a CNN to such high-dimensional text data, which avoids information loss in the word embedding training. Without analyzing the community information, our DeepTip-OH, having three convolution layers in parallel and focusing on the text itself, has the highest precision among all of the studied approaches, because different types of region embedding schemes that complement to each other are employed for higher precision: (1) seq-CNN preserves the order of words that can discriminate patterns of tips from non-tips; and (2) bow-CNN preserves the locality information, i.e., the constituent words heavily contributes to the classification.

With additional unlabeled data, our DeepTip-SEMI outperforms any studied model in terms of F-score and Coverage, as the DeepTip-SEMI learns region embeddings from unlabeled data to predict co-occurrence and absence of words in regions. Furthermore, one hot vectors from text can compensate the potential information loss in embeddings during the training of unlabeled data. In addition, two-view embeddings helps gain more relevant tips at little expense of precision.

VIII. Threats to Validity

The identified threats to validity are listed as follows:

Selection of APIs and Q&A websites. In this paper, we studied the Stack Overflow (SO) posts related to PHP API methods, as PHP is a very popular web development technology and has a large collection of developer-generated SO threads. Therefore, the studied APIs may not be representative of all APIs in other programming languages. However, like the existing studies (e.g., [7], [9]) on APIs, conducting the analysis on one type of APIs is very common. In addition to the selection of APIs, we choose Stack Overflow as it is the largest and most popular developer Q&A website with over 15 million threads. Even though we cannot generalize our findings over other platforms, Stack Overflow has been used as a data source for many other existing API studies (e.g., [1], [7], [48]).

Manual Labeling and user studies. To train a supervised approach, labeled data is necessary. However, it is very hard to find people to help label software data as they need to have some knowledge of PHP programming and also it is very time-consuming. To maximumly reduce the bias, we managed to conduct a pilot exploratory study with three participates to manually identify the sentences or paragraphs that can be used as tips. Through the analysis of our pilot user study, the second author summarizes the principles of labeling data. Like the previous study [1], we recruit another three participants, including the second author of this paper to create a golden dataset with over 31k sentences and 21k paragraphs for tip extraction. Another source of a threat to the validity is human assessment from the 12 participants in our comparative user study like other previous studies [1], [7]. However, we try our best to reduce the threat of individual bias by making sure that every sentence or paragraph is reviewed by three participants.

IX. Related Work

Here, we summarize the some studies relevant to our study.

Mining useful information from technical online resources. An extensive research has been done on mining information from SO posts, such as bug fixes assisting debugging [48], duplicate threads [49], [50], code examples and comments [9], insightful sentences for improving documentation [7], and summarizing opinions of APIs [1], [51]. For example, Campos et al. [48] propose a tool to discover fixes for API-usage-related bugs caused by using API methods or functions incorrectly. The tool locates answer and code snippets to assist debugging. *DupPredictor* by Zhang et al. [49] is an automated tool to detect potential duplicates of a given question by LDA based topic, tags, title, question description similarities. Wong et al. [9] propose a method named AutoComment to mine and leverage code-description mappings from SO to automatically generate description comments for code segments. Some other work leverages crowd knowledge on SO to benefit the usage of integrated development environment (IDE). Ponzanelli et al. [10] and Bacchelli et al. [11] develop Seahawk, an Eclipse plugin integrating text and code snippers into IDE. Seehawk allows users to retrieve Q&A posts from SO, link the related discussions to code samples in Eclipse, and attach explanative comments to the links. Some other studies mine useful information from tutorials [52], [53] and official API documentation [54], [55] Jiang et al. [52], [53] extracts the relevant tutorial segments to a Java class. Robillard et al. [54] extract knowledge items that are relevant to APIs in a block of code from API documentation. However, the above studies are not focusing on the extraction of useful sentences and paragraphs. Treude et al., [55] analyze document corpus and detect task description describing how to accomplish some tasks. To our best knowledge, SISE from Christoph et al. [7] is perhaps most closely related to ours. SISE extracts insightful sentences from SO to augment Java API documentation to improve documentation quality. In this paper, through several experiments, we show that our approaches outperform SISE.

Mining tips from user-generated content. Several researchers conduct studies to extract tip-like information of experiences [56]–[59], and actions or todo items [60]–[62] using semantic attributes and linguistic characteristics. For example, Ryu et al. [61] detect actionable non-imperative clauses using syntactic and modal characteristics. Pareti et al. [62] propose a novel framework for representing community know-how on the Semantic Web. To our best knowledge, three groups claim that their approaches extract tips from user-generated content, like Yahoo! Answers or customer reviews [5], [63], [64]. Weber et al. [63] extract tips from Yahoo! Answers to solve search queries with "how-to" intent and they develop machine learning techniques to eliminate low-quality tips. Our work differs from theirs as their work largely relies on the question-answer structure (e.g. questions starting with "how to", "how could I") but ours employs deep learning to learn semantics. Zhu et al. [64] present an unsupervised method to mine tips from customer reviews. Guy et al. [5] propose to extract and rank travel tips from user-generated reviews. However, none of the above studies are on mining tips from developer Q&A websites. In this paper, we replicate Guy et al.'s approach and our approach can outperform it.

X. Conclusion and Future Work

In this paper, we introduce *DeepTip*, a deep learning based approach using different CNN architectures to extract short, practical, and useful API tips for solving a specific question or learning APIs in general from a large collection of Stack Overflow (SO) posts. Our empirical results show that DeepTip-SEMI, using a parallel CNN architecture and integrating unlabeled data, is effective in tip extraction. More importantly, the feedback from real SO users in a user study indicates that tip extraction is very useful to them and *DeepTip* can extract more useful tips than a state-of-the-art approach.

In the future, we plan to apply DeepTip to other programming languages and invite more participants with different backgrounds to share their feedback.

References

[1] G. Uddin and F. Khomh, "Automatic summarization of api reviews," in *Automated Software Engineering (ASE), 2017 32nd IEEE/ACM International Conference on*. IEEE, 2017, pp. 159–170.

[2] G. Petrosyan, M. P. Robillard, and R. De Mori, "Discovering information explaining api types using text classification," in *Software Engineering (ICSE), 2015 IEEE/ACM 37th IEEE International Conference on*, vol. 1. IEEE, 2015, pp. 869–879.

[3] L. Mamykina, B. Manoim, M. Mittal, G. Hripcsak, and B. Hartmann, "Design lessons from the fastest q&a site in the west," in *Proceedings of the SIGCHI conference on Human factors in computing systems*. ACM, 2011, pp. 2857–2866.

[4] A. Barua, S. W. Thomas, and A. E. Hassan, "What are developers talking about? an analysis of topics and trends in stack overflow," *Empirical Software Engineering*, vol. 19, no. 3, pp. 619–654, 2014.

[5] I. Guy, A. Mejer, A. Nus, and F. Raiber, "Extracting and ranking travel tips from user-generated reviews," in *Proceedings of the 26th international conference on world wide web*. International World Wide Web Conferences Steering Committee, 2017, pp. 987–996.

[6] M. Ahasanuzzaman, M. Asaduzzaman, C. K. Roy, and K. A. Schneider, "Mining duplicate questions of stack overflow," in *Mining Software Repositories (MSR), 2016 IEEE/ACM 13th Working Conference on*. IEEE, 2016, pp. 402–412.

[7] C. Treude and M. P. Robillard, "Augmenting api documentation with insights from stack overflow," in *Software Engineering (ICSE), 2016 IEEE/ACM 38th International Conference on*. IEEE, 2016, pp. 392–403.

[8] J. Saxe, D. Mentis, and C. Greamo, "Mining web technical discussions to identify malware capabilities," in *Distributed Computing Systems Workshops (ICDCSW), 2013 IEEE 33rd International Conference on*. IEEE, 2013, pp. 1–5.

[9] E. Wong, J. Yang, and L. Tan, "Autocomment: Mining question and answer sites for automatic comment generation," in *Automated Software Engineering (ASE), 2013 IEEE/ACM 28th International Conference on*. IEEE, 2013, pp. 562–567.

[10] L. Ponzanelli, A. Bacchelli, and M. Lanza, "Seahawk: Stack overflow in the ide," in *Software Engineering (ICSE), 2013 35th International Conference on*. IEEE, 2013, pp. 1295–1298.

[11] A. Bacchelli, L. Ponzanelli, and M. Lanza, "Harnessing stack overflow for the ide," in *Proceedings of the Third International Workshop on Recommendation Systems for Software Engineering*. IEEE Press, 2012, pp. 26–30.

[12] Y. Kim, "Convolutional neural networks for sentence classification," *arXiv preprint arXiv:1408.5882*, 2014.

[13] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Advances in neural information processing systems*, 2013, pp. 3111–3119.

[14] C. M. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2006.

[15] G. E. Hinton, "Learning distributed representations of concepts," in *Proceedings of the eighth annual conference of the cognitive science society*, vol. 1. Amherst, MA, 1986, p. 12.

[16] Y. Bengio, R. Ducharme, P. Vincent, and C. Jauvin, "A neural probabilistic language model," *Journal of machine learning research*, vol. 3, no. Feb, pp. 1137–1155, 2003.

[17] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *arXiv preprint arXiv:1301.3781*, 2013.

[18] J. Pennington, R. Socher, and C. Manning, "Glove: Global vectors for word representation," in *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, 2014, pp. 1532–1543.

[19] A. Joulin, E. Grave, P. Bojanowski, and T. Mikolov, "Bag of tricks for efficient text classification," *arXiv preprint arXiv:1607.01759*, 2016.

[20] R. Johnson and T. Zhang, "Effective use of word order for text categorization with convolutional neural networks," *arXiv preprint arXiv:1412.1058*, 2014.

[21] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[22] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa, "Natural language processing (almost) from scratch," *Journal of Machine Learning Research*, vol. 12, no. Aug, pp. 2493–2537, 2011.

[23] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," *arXiv preprint arXiv:1207.0580*, 2012.

[24] S. Bird and E. Loper, "Nltk: the natural language toolkit," in *Proceedings of the ACL 2004 on Interactive poster and demonstration sessions*. Association for Computational Linguistics, 2004, p. 31.

[25] G. G. Chowdhury, *Introduction to modern information retrieval*. Facet publishing, 2010.

[26] D. Movshovitz-Attias, Y. Movshovitz-Attias, P. Steenkiste, and C. Faloutsos, "Analysis of the reputation system and user contributions on a question answering website: Stackoverflow," in *Proceedings of the 2013 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining*. ACM, 2013, pp. 886–893.

[27] Q. Tian, P. Zhang, and B. Li, "Towards predicting the best answers in community-based question-answering services." in *ICWSM*, 2013.

[28] F. Calefato, F. Lanubile, M. C. Marasciulo, and N. Novielli, "Mining successful answers in stack overflow," in *Mining Software Repositories (MSR), 2015 IEEE/ACM 12th Working Conference on*. IEEE, 2015, pp. 430–433.

[29] R. Johnson and T. Zhang, "Semi-supervised convolutional neural networks for text categorization via region embedding," in *Advances in neural information processing systems*, 2015, pp. 919–927.

[30] R. K. Ando and T. Zhang, "A framework for learning predictive structures from multiple tasks and unlabeled data," *Journal of Machine Learning Research*, vol. 6, no. Nov, pp. 1817–1853, 2005.

[31] ——, "Two-view feature generation model for semi-supervised learning," in *Proceedings of the 24th international conference on Machine learning*. ACM, 2007, pp. 25–32.

[32] G. H. John and P. Langley, "Estimating continuous distributions in bayesian classifiers," in *Proceedings of the Eleventh conference on Uncertainty in artificial intelligence*. Morgan Kaufmann Publishers Inc., 1995, pp. 338–345.

[33] J. Fogarty, R. S. Baker, and S. E. Hudson, "Case studies in the use of roc curve analysis for sensor-based estimates in human computer interaction," in *Proceedings of Graphics Interface 2005*. Canadian Human-Computer Communications Society, 2005, pp. 129–136.

[34] D. W. Aha, D. Kibler, and M. K. Albert, "Instance-based learning algorithms," *Machine learning*, vol. 6, no. 1, pp. 37–66, 1991.

[35] J. R. Quinlan, *C4. 5: programs for machine learning*. Elsevier, 2014.

[36] L. Breiman, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

[37] M. M. Adankon and M. Cheriet, "Support vector machine," in *Encyclopedia of biometrics*. Springer, 2009, pp. 1303–1308.

[38] J. Friedman, T. Hastie, and R. Tibshirani, *The elements of statistical learning*. Springer series in statistics New York, 2001, vol. 1.

[39] K. Crammer, A. Kulesza, and M. Dredze, "Adaptive regularization of weight vectors," in *Advances in neural information processing systems*, 2009, pp. 414–422.

[40] J. Leskovec, A. Rajaraman, and J. D. Ullman, *Mining of massive datasets*. Cambridge university press, 2014.

[41] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The weka data mining software: an update," *ACM SIGKDD explorations newsletter*, vol. 11, no. 1, pp. 10–18, 2009.

[42] Y. Yang and J. O. Pedersen, "A comparative study on feature selection in text categorization," in *Icml*, vol. 97, 1997, pp. 412–420.

[43] L. Kotthoff, C. Thornton, H. H. Hoos, F. Hutter, and K. Leyton-Brown, "Auto-weka 2.0: Automatic model selection and hyperparameter optimization in weka," *The Journal of Machine Learning Research*, vol. 18, no. 1, pp. 826–830, 2017.

[44] Y. Wu, S. C. Hoi, and N. Yu, "Libsol: A library for scalable online learning algorithms," *SMU Technical Report (SMU-TR-2016-07-25)*, 2016.

[45] R. Rehurek and P. Sojka. (2008) Gensim - top modeling for humans. [Online]. Available: https://radimrehurek.com/gensim/

[46] G. Zhao and J. Huang, "Deepsim: deep learning code functional similarity," in *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. ACM, 2018, pp. 141–151.

[47] C. H. Stewart Butterfield, Eric Costello and S. Mourachov. (2013) slack: Where work happens. [Online]. Available: https://slack.com/

[48] E. C. Campos, M. Monperrus, and M. A. Maia, "Searching stack overflow for api-usage-related bug fixes using snippet-based queries," in *Proceedings of the 26th Annual International Conference on Computer Science and Software Engineering*. IBM Corp., 2016, pp. 232–242.

[49] Y. Zhang, D. Lo, X. Xia, and J.-L. Sun, "Multi-factor duplicate question detection in stack overflow," *Journal of Computer Science and Technology*, vol. 30, no. 5, pp. 981–997, 2015.

[50] M. Ahasanuzzaman, M. Asaduzzaman, C. K. Roy, and K. A. Schneider, "Mining duplicate questions in stack overflow," in *Proceedings of the 13th International Conference on Mining Software Repositories*, ser. MSR '16. New York, NY, USA: ACM, 2016, pp. 402–412. [Online]. Available: http://doi.acm.org/10.1145/2901739.2901770

[51] G. Uddin and F. Khomh, "Mining aspects in api reviews," *Polytechnique Montréal, Tech. Rep.*, 2017.

[52] 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*. IEEE Press, 2017, pp. 38–48.

[53] H. Jiang, J. Zhang, X. Li, Z. Ren, and D. Lo, "A more accurate model for finding tutorial segments explaining apis," in *Software Analysis, Evolution, and Reengineering (SANER), 2016 IEEE 23rd International Conference on*, vol. 1. IEEE, 2016, pp. 157–167.

[54] M. P. Robillard and Y. B. Chhetri, "Recommending reference api documentation," *Empirical Software Engineering*, vol. 20, no. 6, pp. 1558–1586, 2015.

[55] C. Treude, M. P. Robillard, and B. Dagenais, "Extracting development tasks to navigate software documentation," *IEEE Transactions on Software Engineering*, vol. 41, no. 6, pp. 565–581, 2015.

[56] K. C. Park, Y. Jeong, and S. H. Myaeng, "Detecting experiences from weblogs," in *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, 2010, pp. 1464–1472.

[57] H.-J. Min and J. C. Park, "Identifying helpful reviews based on customers mentions about experiences," *Expert Systems with Applications*, vol. 39, no. 15, pp. 11 830–11 838, 2012.

[58] Q. Nguyen, "Detecting experience revealing sentences in product reviews," *University of Amsterdam*, 2012.

[59] C. S. Sauer and T. Roth-Berghofer, "Solution mining for specific contextualised problems: towards an approach for experience mining," in *Proceedings of the 21st International Conference on World Wide Web*. ACM, 2012, pp. 729–738.

[60] K. Nagao, K. Inoue, N. Morita, and S. Matsubara, "Automatic extraction of task statements from structured meeting content," in *Knowledge Discovery, Knowledge Engineering and Knowledge Management (IC3K), 2015 7th International Joint Conference on*, vol. 1. IEEE, 2015, pp. 307–315.

[61] J. Ryu, Y. Jung, and S.-H. Myaeng, "Actionable clause detection from non-imperative sentences in howto instructions: A step for actionable information extraction," in *International Conference on Text, Speech and Dialogue*. Springer, 2012, pp. 272–281.

[62] P. Pareti, E. Klein, and A. Barker, "A semantic web of know-how: linked data for community-centric tasks," in *Proceedings of the 23rd International Conference on World Wide Web*. ACM, 2014, pp. 1011–1016.

[63] I. Weber, A. Ukkonen, and A. Gionis, "Answers, not links: extracting tips from yahoo! answers to address how-to web queries," in *Proceedings of the fifth ACM international conference on Web search and data mining*. ACM, 2012, pp. 613–622.

[64] D. Zhu, T. Lappas, and J. Zhang, "Unsupervised tip-mining from customer reviews," *Decision Support Systems*, vol. 107, pp. 116–124, 2018.