# Post2Vec: Learning Distributed Representations of Stack Overflow Posts

Bowen Xu, Thong Hoang, Abhishek Sharma, Chengran Yang, Xin Xia, and David Lo

**Abstract**—Past studies have proposed solutions that analyze Stack Overflow content to help users find desired information or aid various downstream software engineering tasks. A common step performed by those solutions is to extract suitable representations of posts; typically, in the form of meaningful vectors. These vectors are then used for different tasks, for example, tag recommendation, relatedness prediction, post classification, and API recommendation. Intuitively, the quality of the vector representations of posts determines the effectiveness of the solutions in performing the respective tasks. In this work, to aid existing studies that analyze Stack Overflow posts, we propose a specialized deep learning architecture Post2Vec which extracts distributed representations of Stack Overflow posts. Post2Vec is aware of different types of content present in Stack Overflow posts, i.e., title, description, and code snippets, and integrates them seamlessly to learn post representations. Tags provided by Stack Overflow users that serve as a common vocabulary that captures the semantics of posts are used to guide Post2Vec in its task. To evaluate the quality of Post2Vec's deep learning architecture, we first investigate its end-to-end effectiveness in tag recommendation task. The results are compared to those of state-of-the-art tag recommendation approaches that also employ deep neural networks. We observe that Post2Vec achieves 15-25 percent improvement in terms of F1-score@5 at a lower computational cost. Moreover, to evaluate the value of representations learned by Post2Vec, we use them for three other tasks, i.e., relatedness prediction, post classification, and API recommendation. We demonstrate that the representations can be used to boost the effectiveness of state-of-the-art solutions for the three tasks by substantial margins (by 10, 7, and 10 percent in terms of F1-score, F1-score, and correctness, respectively). We release our replication package at https://github.com/maxxbw/Post2Vec.

## 1 Introduction

Presently, software question and answer (SQA) sites are an essential part of developers' day-to-day work for problem-solving and self-learning. A SQA site (such as Stack Overflow) is a collection of posts where a post contains several components, e.g., title, body, and tags as illustrated in Fig. 1. To help developers navigate SQA sites and find relevant information efficiently, many solutions have been proposed. They address concrete tasks such as recommending tags to posts (aka. tag recommendation) [1], [2], [3], identification of related posts (aka. relatedness prediction) [4], [5], [6], [7], post classification [8], [9], [10], and many more. A common step in previous works is to extract a suitable vector to represent a post for further processing. Intuitively, following "garbage in, garbage out" principle, the quality of a post's vector representation plays a major role in determining the eventual outcomes.

In this work, to boost the effectiveness of existing solutions that analyze SQA site contents, we want to learn distributed representations of posts that can be used for a number of downstream tasks. We propose a new deep learning architecture named Post2Vec that can effectively embed posts in a space where the distance between similar posts is small. Specifically, considering the nature of tags, i.e., semantic labels provided by developers and shared by many posts, we utilize them to supervise the learning of post representations.

Intuitively, this problem is challenging because it requires learning a correspondence between the entire content of a post and only a few semantic labels among hundreds of thousands of candidates. That is, it requires associating contents from different components of a post, which typically include hundreds of words and several pieces of code snippets, into a few descriptive labels. Moreover, to develop a robust and useful post representation, Post2Vec needs to address the following three limitations of prior works:

- Posts from a particular SQA site typically follow a standard structure, which consists of different components (e.g., title and body) carrying information at different levels of detail. The title summarizes a specific problem, while the body expands the title with more details. However, we found that many prior studies, e.g., [3], [4], simply treated the different components equally and represented a post by concatenating the content of the various components together into one single piece of text.
- Many previous works, e.g., [11], [12], [13], have removed code snippets during the preprocessing step due to these code snippets are short, have poor structure, and are written in different programming languages. However, from Fig. 1 we can see that if

- *Bowen Xu, Thong Hoang, Abhishek Sharma, Chengran Yang, and David Lo are with the School of Computing and Information Systems, Singapore Management University, Singapore 188065, Singapore. E-mail: {bowenxu.2017, vdthoang.2016, abhisheksh.2014, cryang, davidlo}@smu.edu.sg.*
- *Xin Xia is with Software Engineering Application Technology Lab, Huawei, Shenzhen, Guangdong 518129, China. E-mail: xin.xia@acm.org.*

*Manuscript received 16 Mar. 2021; revised 21 June 2021; accepted 21 June 2021. Date of publication 1 July 2021; date of current version 19 Sept. 2022. (Corresponding author: Bowen Xu.) Recommended for acceptance by E. Murphy-Hill. Digital Object Identifier no. 10.1109/TSE.2021.3093761*

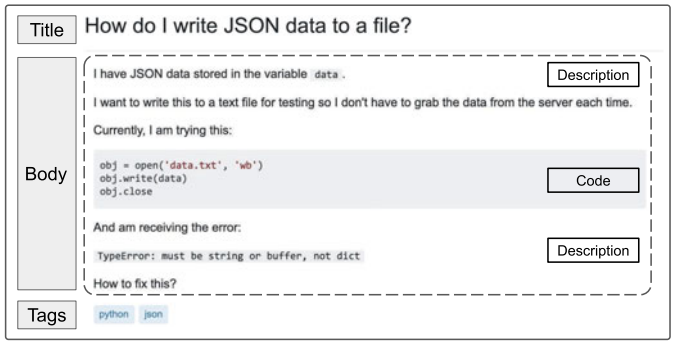



Fig. 1. Example of a stack overflow post ID=12309269.

we consider the code snippets, it becomes effortless to identify that the post is *Python* related even though the user does not mention Python in the title and description. Thus, intuitively, code snippets should be considered while learning an effective post representation. Moreover, we found that for more than 70 percent of posts in the Stack Overflow data dump (dated September 5, 2018), their body contained at least one code snippet. In other words, code snippets are vital sources of information if they can be properly captured while learning representations of posts.

- Early attempts [1], [2] focused only on extracting text from posts and then used the text to create well known but simplistic representations such as bag-of-words (BOW). A BOW representation of a post is a multi-set of words that appear in the post. However, BOW ignores the semantic relation between words.

Post2Vec uses Word2Vec [14] to map words with similar meaning to vectors with small distances between them. Additionally, Post2Vec leverages not only the natural language content (i.e., title and description) presented in the post but also the code snippets. Individually, title, description, and code snippets are considered as three components and they are fed into three different neural network models to produce three fixed length feature vectors. Finally, considering the nature of tags, we utilize them to supervise the learning of post representations from the three feature vectors. More specifically, Post2Vec optimizes the vector representations of posts to predict appropriate tags that are assigned to the respective posts by learning the relationship between the vectors and tags.

Once the post representations are learned, they can be used to boost the effectiveness of many supervised learning tasks, especially those rely only on a limited set of labeled data. Although Stack Overflow data is vast, many prior studies only made use of a subset of posts that are labeled [4], [10]. By using the learned post representations, in effect, we are using the power of vast unlabeled data to help in a particular supervised learning task (aka. semi-supervised learning). Additionally, many prior approaches, e.g., [6], [15], proposed handcrafted features for a specific task and represented a post by a low dimensional feature vector. The effectiveness of those approaches may benefit from the post representations that are pre-trained based on a large dataset. Such representations can be used to embellish the low dimensional feature vector to boost effectiveness potentially.

The closest work to ours is by Zhou *et al*. [3], who proposed a deep neural network architecture to predict tags that are assigned to SQA posts. Our work is different from them in the following aspects. First, the main goal of our work and theirs differ; Zhou *et al*. aim to improve tag recommendation, while we aim to produce representations of posts that can be used for multiple downstream tasks. Second, the components considered in our work and theirs differ; Zhou *et al*. only utilize title and description while we consider code snippets as well. Consider many tasks (e.g., tag prediction [2], [3], similar post detection [5], [6], etc) are expected to be performed as soon as the post is created, hence our approach is designed to construct vector representations of newly created posts, i.e., those without answers and comments. With this design, we can immediately infer the vector representations of posts once they are created, and directly use the representations for other tasks. Third, the way the components are processed differ too; Zhou *et al*. concatenate title and description to a piece of text in the data preprocessing step while we feed different components to different models separately. Another closely related work is Doc2Vec [16] that generates a vector representation for textual documents; however, there are significant differences between "flat" textual documents considered in Doc2Vec and posts. We consider a post as a structured document with three components (i.e., title, description, and code snippets). Moreover, different from Doc2Vec, we use tags as guides in the generation of post representations.

To evaluate Post2Vec, we first investigate the effectiveness of our approach in the tag recommendation task and compare it with the state-of-the-art approach [3]. Post2Vec achieves 15-25 percent improvement in terms of F1-score@5. Moreover, we further evaluate the value of the representation learned by Post2Vec in three other tasks, i.e., relatedness prediction [4], post classification [10], and API recommendation [17]. We found that these state-of-the-art approaches are improved by integrating our post representation, i.e., 10, 7, and 10 percent for relatedness prediction, post classification, and API recommendation, respectively.

The main contributions of this work are as follows:

- We propose a specialized deep learning architecture Post2Vec that learns vector representations of SQA posts. To the best of our knowledge, this work is the first to explore the learning of generic representation of SQA posts. We focus on Stack Overflow posts, but the solution presented here can be easily adapted for posts from other SQA sites.
- We empirically compare end-to-end Post2Vec with the state-of-the-art neural network based approach for tag recommendation task based on a 10 million posts dataset, and demonstrate that Post2Vec can achieve 15-25 percent improvements and also complete the learning process by up to 1.5 days faster.
- We empirically investigate the benefit of design decision from three aspects. For each aspect, we implement and compare pairs of Post2Vec variants. The experimental results show that, (1) CNN-based Post2Vec outperform LSTM-based Post2Vec significantly and consumes fewer computing resources

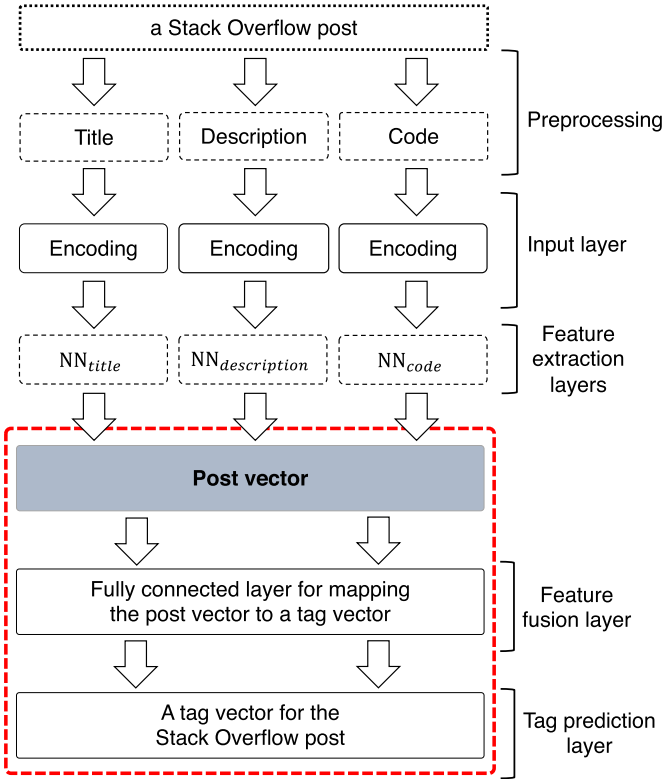



Fig. 2. The overall framework of Post2Vec. Neural network architecture is used to construct the embedding vectors for title, description, and code snippets included in a Stack Overflow post. These embedding vectors are then concatenated to build a vector representation for the post (post vector). The post vector is learned by minimizing an objective function of the tag prediction layer.

during model training, (2) consideration of code snippets can boost effectiveness but consumes more computing resources during model training, (3) considers different post components separately significantly boosts the performance but consumes more computing resources than considers different components together during model training.

- We empirically investigate the value of integrating the post vectors generated by Post2Vec and feature vectors used by the state-of-the-art approaches of three tasks (relatedness prediction, post classification, and API recommendation) and demonstrate substantial improvements.

The rest of this paper is organized as follows. Section 2 elaborates the design of our approach Post2Vec. Section 3 presents our experiments that compare Post2Vec with the state-of-the-art tag recommendation approach and their results. Section 4 investigates the value of our learned post representations to aid in the three downstream tasks (relatedness prediction, post classification, and API recommendation). Section 5 presents a qualitative study and some threats to validity. Section 6 describes related prior studies. We conclude and mention future work in Section 7.

## 2 Proposed Approach

### 2.1 Framework Overview

Fig. 2 illustrates the overall framework of Post2Vec that takes Stack Overflow (SO) posts as input and outputs their distributed representations in the form of post vectors. More specifically, Post2Vec consists of four parts:

- *Preprocessing*: This part extracts three pieces of information from a SO post (i.e., title, description, and code snippets), cleans them, and represents each of them as a sequence of tokens.
- *Input layer*: This part encodes the sequences of tokens corresponding to the title, description, and code snippets of posts into two-dimensional matrices as inputs to the NNs in the feature extraction layers.
- *Feature extraction layers*: This part extracts the embedding vectors (aka. features) of the title, description, and code snippets. These embedding vectors are then concatenated to form the post vector representation of the input SO post.
- *Feature fusion and tag prediction layers*: This part maps the post vector to a tag vector; the tag vector indicates the probabilities of various tags being assigned to the SO post.

Post2Vec leverages the tags assigned to SO posts to guide the learning of suitable post vectors. And then, the learned vectors are used as the distributed representations of SO posts. We use tags as a guide since they are semantic labels provided by developers and shared by many posts. Specifically, we define a learning task to construct prediction function $\mathbf{f} : P \rightarrow \mathcal{Y}$, where $y_i \in \mathcal{Y}$ indicates a list of tags of the post $p_i \in P$. The prediction function $\mathbf{f}$ is learned by minimizing the differences between predicted and actual tags assigned to posts. After the prediction function $\mathbf{f}$ is learned, for each post, we can obtain its post vector from the intermediate output between the feature extraction and feature fusion layers (see Fig. 2). We explain the details of each part of Post2Vec in the following subsections.

### 2.2 Preprocessing

We process the title, description, and code snippets of each post by the following four steps that are applied one after the other:

(1) *Separate description and code snippets from the body*. Code snippets in the body of a post are enclosed in HTML tags ⟨pre⟩⟨code⟩ and ⟨\code⟩⟨\pre⟩ [4], [15]. Hence, we use a regular expression "*⟨pre⟩⟨code⟩([\s\S]*?)⟨\cr code⟩⟨\cr pre⟩*" to extract those code snippets. We concatenate all code snippets in a post into one document, and the remaining parts of the post into another document.

(2) *Remove HTML tags*. We remove HTML tags that appear in the extracted code snippets and description documents. These HTML tags typically correspond to formatting instructions that may not carry much semantic meaning [4]. To achieve this, we utilize a popular HTML parser Beautiful Soup.[1]

(3) *Tokenize title, description, and code snippets*. At this step, we would like to split each title, description, and code snippet into a sequence of tokens. To do

1. Beautiful Soup, https://www.crummy.com/software/BeautifulSoup.

this step, we employ a popular tool named NLTK.[2] More precisely, the function *word_tokenize* was used in our implementation.

(4) *Construct component-specific vocabulary*. Based on the posts in training data, we build three vocabularies $\mathcal{V}^{\mathrm{T}}$, $\mathcal{V}^{\mathrm{D}}$, $\mathcal{V}^{\mathrm{C}}$; each vocabulary corresponds to a set of tokens along with their occurrence frequencies in the collection of title, description, and code snippet, respectively. A common practice for constructing a vocabulary from large scale data is to discard the tokens that occur less than $k$ times [14]. In this work, we set $k$ equals to 50 which is a common threshold used in prior works, e.g., [18].

At the end of the preprocessing steps, title, description, and code snippets of each post are extracted as three *sequences of tokens*. These sequences are fed to the input layer of our proposed framework for further processing.

### 2.3 Input Layer

The input layer performs three main steps:

(1) *Indexing and padding*. We convert each component of a post output by the preprocessing step (which is a sequence of tokens of arbitrary length) to a *fixed length* sequence of *token indices*. To achieve this, we first replace each token in each sequence by its index in the corresponding vocabulary (if it exists in the vocabulary; otherwise we skip the token). For example, given a title as a sequence of tokens $\mathrm{T} = [w_1, \ldots, w_{|\mathrm{T}|}]$, we convert it to $[index_1, \ldots, index_{|\mathrm{T}|}]$, where $|\mathrm{T}|$ is a length of the sequence of tokens and $index_i$ is the index of $w_i$ in $\mathcal{V}^{\mathrm{T}}$. Since components of different posts contain different numbers of tokens, and NN often requires each input to be represented in the same way for the purpose of parallelization, we perform padding or truncating which is a standard solution. Specifically, we set three parameters $\mathcal{L}^T$, $\mathcal{L}^D$, and $\mathcal{L}^C$ to be the target length of the sequence of tokens in the title, description, and code snippets respectively. For each sequence belonging to a component (i.e., title, description, or code snippets), if its length is smaller than the predefined length (i.e., $\mathcal{L}^T$, $\mathcal{L}^D$, or $\mathcal{L}^C$), we add a special token[3] (one or more times) so that all sequences have the same length. If the length of a sequence belonging to a component is longer than the predefined length, we truncate it to only its first $\mathcal{L}^T$, $\mathcal{L}^D$, or $\mathcal{L}^C$ tokens. We calculate the distribution of the number of tokens from each component among all posts. And we found that by setting the value of $\mathcal{L}^T$, $\mathcal{L}^D$, $\mathcal{L}^C$ as 100, 1,000, and 1,000, we can preserve more than 95 percent of the original tokens for each component. In other words, more than 95 percent of posts' title, description, and code length are smaller than 100, 1,000, and, 1,000. Thus only a minimal loss of data is incurred while keeping the representation to a smaller size, for a boost in performance (esp. training time).

(2) *Token representation*. We represent each token as a vector of $d$ values. By default, we set $d$ to be 128. We initialize the $d$ values randomly and they will be updated and learned during the training process.

(3) *Component representation*. We encode the title, description, and code snippets of a post as two-dimensional matrices (i.e., $\mathcal{X}^{\mathrm{T}}$, $\mathcal{X}^{\mathrm{D}}$, and $\mathcal{X}^{\mathrm{C}}$). For example, given a title as a sequence of tokens T, we construct its matrix representation $\mathcal{X}^{\mathrm{T}} \in \mathbb{R}^{\mathcal{L}^T \times d}$, where $d$ is the dimension of the token representation. At the end, the input layer outputs $\mathcal{X}^{\mathrm{T}}$, $\mathcal{X}^{\mathrm{D}}$, and $\mathcal{X}^{\mathrm{C}}$ and these matrices are used as an input to the feature extraction layers.

### 2.4 Feature Extraction Layers

Under the framework of Post2Vec, we utilize NN models to extract feature from posts. Specifically, we input $\mathcal{X}^{\mathrm{T}}$, $\mathcal{X}^{\mathrm{D}}$, and $\mathcal{X}^{\mathrm{C}}$, which are the encoded data of the title, description, and code snippets, for each Stack Overflow (SO) post to three independent NN models. The NN models produce embedding vectors $\mathbf{z_T}$, $\mathbf{z_D}$, and $\mathbf{z_C}$, which represent the features extracted from each component of the SO post, respectively. Considering code snippets in Stack Overflow are short, have poor structure, and are written in different programming languages [11], [12], [13], the existing code representation techniques are unable to be applied as they are applicable for either a single type of program language (e.g., [19], [20]) or the code should be at least a complete function or method (e.g., [20], [21]). Hence, Post2Vec processes code snippets and other two textual components (i.e., title and description) in the same way. Noted that different types of NN models can be used as feature extractors. By default, we use CNN as the feature extractor. The reason is that CNN is considered a promising solution for a classification task when class labels are usually determined by some key phrases [22]. In our case (i.e., in the context of software question and answer data), it is reasonable to choose CNN since the semantics of posts are usually determined by some key phrases containing technical terms. For example, CNN can effectively identify the posts containing key phrases like "throws IOException" as Java-related and recommend the tag 'Java' for them. Moreover, we also investigate the performance of another widely used NN model, i.e., LSTM, in our experiment. We describe the hyperparameters of the two considered NNs in Section 3.3 and the result of their comparison in Section 3.6.

### 2.5 Feature Fusion and Tag Prediction Layers

Fig. 3 shows the details of the part of the architecture shown inside the red dashed box in Fig. 2. The inputs of this part are the three embedding vectors (i.e., $\mathbf{z_T}$, $\mathbf{z_D}$, and $\mathbf{z_C}$) which represent the features extracted from the title, description, and code snippets respectively. These embedding vectors are concatenated to construct a new embedding vector representing a Stack Overflow post.

We put the embedding vector $\mathbf{z}$ into a fully connected (FC) layer to produce a vector $\mathbf{h}$:

$$\mathbf{h} = \alpha(\mathbf{w_h} \cdot \mathbf{z} + b_{\mathbf{h}}), \tag{1}$$

2. NLTK Tokenizer, https://www.nltk.org/api/nltk.tokenize.html.
3. We use zero as the padded token.

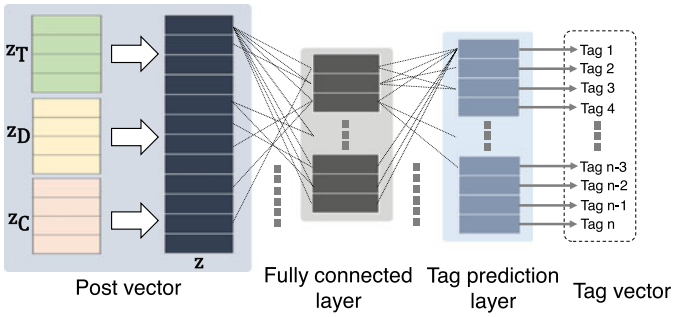



Fig. 3. The details of the red dashed box in Fig. 2. $\mathbf{z_T}$, $\mathbf{z_D}$, and $\mathbf{z_C}$ are the embedding vectors of the title, description, and code snippet, respectively. $\mathbf{z}$ is the post vector and is fed to a fully-connected layer (FC) to produce the tag vector (i.e., the probability distribution over tags).

where $\mathbf{w_h}$ is the weight matrix used to connect the embedding vector $\mathbf{z}$ with the FC layer, $\cdot$ is the dot product between $\mathbf{w_h}$ and $\mathbf{z}$, and $b_\mathbf{h}$ is the bias value. Finally, the vector $\mathbf{h}$ is passed to a tag prediction layer to produce the following:

$$\mathbf{t} = -\mathbf{h} \cdot \mathbf{w_t}, \tag{2}$$

where $\mathbf{w_t}$ is the weight matrix between the FC layer and the tag prediction layer, and $\mathbf{t} \in \mathbb{R}^{\mathbb{T}\times 1}$ ($\mathbb{T}$ is a total number of tags). We then apply the sigmoid function [23] to get the probability distribution over tags as follows:

$$\mathrm{p}(t_i|p_i) = \frac{exp(t_i)}{exp(t_i)+1}, \tag{3}$$

where $t_i \in \mathbf{t}$ is the score of the $i^{th}$ tag and $p_i$ is the post for which we want to recommend tags to.

### 2.6 Parameters Learning

Post2Vec aims to learn the following parameters: the embedding matrices of the title, description, and code snippets of each SO post, the convolutional layers' matrices, and the weights matrices and bias values of the fully connected layer and the tag prediction layer. After these parameters are learned, the post vector of each SO post can be determined. These parameters of Post2Vec are learned by minimizing the following objective function:

$$\begin{aligned}\mathcal{O} = &\sum_{y_i \in \mathbf{y}} y_i \times (-\log(\mathrm{p}(t_i|p_i))) + (1-y_i) \\ &\times (-\log(1-\mathrm{p}(t_i|p_i))) + \frac{\lambda}{2}\|\theta\|_2^2,\end{aligned} \tag{4}$$

where $\mathrm{p}(t_i|p_i)$ is the predicted tag probability defined in Equation (3), $y_i = \{0,1\}$ indicates whether the $i^{th}$ tag is assigned to the post $p_i$, and $\theta$ contains all parameters our model. The term $\frac{\lambda}{2}\|\theta\|_2^2$ is used to prevent overfitting in the training process [24]. The dropout technique [25] is employed to improve the robustness of Post2Vec. We use Adam [26] to minimize the objective function since Adam has been shown to be computationally efficient and it requires low memory consumption as compared to other optimization techniques [26], [27], [28]. We also use back-propagation [29], a simple implementation of the chain rule of partial derivatives, to efficiently compute parameter updates during the training process.

## 3 Post2Vec for Tag Recommendation

To evaluate Post2Vec's performance, we investigate its performance for tag recommendation task. The problem formulation of tag recommendation task is as follows: given a set of existing software posts that are labeled with tags, how to automatically predict a set of appropriate tags for a new unseen software post. Arguably, if Post2Vec can learn good embeddings, its end-to-end performance for the task of learning suitable tags to be assigned to Stack Overflow posts should be high. In this section, we conduct experiments to mainly answer the following three research questions:

*RQ1: Compared with the state-of-the-art approach, how effective and efficient is Post2Vec in tag recommendation?*

Recently, Zhou *et al.* [3] proposed deep learning based tag recommendation approaches. To demonstrate the value of Post2Vec for tag recommendation task, we compare our approach with theirs.

*RQ2: What is the impact of different types of NNs as feature extractors?*

One of the novelties of Post2Vec over the state-of-the-art tag recommendation approaches is the NNs used for extracting features can be replaced by other types. Here, we want to investigate the impact of different types of NNs on the performance of Post2Vec.

*RQ3: Does Post2Vec benefit from code snippets?*

One novelty of Post2Vec over the state-of-the-art tag recommendation approaches is the consideration of code snippets. Considering the fact that code snippets in Stack Overflow are incomplete, have poor structure, and are written in different programming languages [30], it is unclear if their consideration can be a net benefit to Post2Vec. Furthermore, many previous works, e.g., [11], [13], remove the code snippets when analyzing posts. In this research question, we want to investigate if code snippets boost (or harm) Post2Vec performance.

*RQ4: What is the impact of handling post components separately rather than combining them?*

Another novelty of Post2Vec over the state-of-the-art tag recommendation approaches is the deep learning architecture that considers different post components (title, description, and code snippets) separately using three different NN models (i.e., $\mathrm{NN}_{title}$, $\mathrm{NN}_{description}$, and $\mathrm{NN}_{code}$). Here, we would like to investigate if handling post components separately can boost the performance of Post2Vec.

### 3.1 Baseline Approaches

To answer RQ1, we compare the Post2Vec with the state-of-the-art tag recommendation approach proposed by Zhou *et al.* [3]. In particular, they proposed four types of neural network based approaches, TagCNN, TagRNN, TagHAN, and TagRCNN, which were based on convolutional neural network (CNN) [31], recurrent neural network (RNN) [32], hierarchical attention network (HAN) [33], and recurrent convolutional neural network (RCNN) [34], respectively. Moreover, they also compared the four proposed approaches with the three traditional approaches, EnTagRec [35], TagMulRec [36], and FastTagRec [37]. Their experimental results showed that TagCNN and TagRCNN were the top-2 best performers in terms of effectiveness. Moreover, Zhou *et al.* also concluded that TagCNN and

TABLE 1
Differences Between Variants of Post2Vec

| Approach | Used NN Model | Considered Components | | | Architecture |
|---|---|---|---|---|---|
| | | Title | Description | Code Snippets | |
| Post2Vec$_{LSTM,All,Sep}$ | BiLSTM | ✓ | ✓ | ✓ | Multiple LSTMs for different components |
| Post2Vec$_{CNN,-Code,Sep}$ | CNN | ✓ | ✓ | | Multiple CNNs for different components |
| Post2Vec$_{CNN,All,Com}$ | CNN | ✓ | ✓ | ✓ | Single CNN model |
| Post2Vec$_{CNN,All,Sep}$ | CNN | ✓ | ✓ | ✓ | Multiple CNNs for different components |

TagRCNN are practical for use on the various-scale datasets. Thus, we use both TagCNN and TagRCNN as our baseline approaches. To replicate Zhou *et al.*'s approach, we carefully use their released source code.[4]

Moreover, we raise three research questions (i.e., RQ2, RQ3, RQ4) evaluating the impact of Post2Vec design decisions: RQ2 investigates the impact of using different types of NN models in Post2Vec. RQ3 investigates the impact of inclusion or exclusion of code snippets in the post components considered by Post2Vec. RQ4 investigates the impact of concatenating or separating different post components. To answer these three research questions, we implement four variants of Post2Vec. Table 1 summarizes the differences between all investigated variants considering three perspectives: used NN models, used components, and architecture. To answer RQ2, we compare the performance of pairs of variants that are only different on used NN models, i.e., Post2Vec$_{LSTM,All,Sep}$ and Post2Vec$_{CNN,All,Sep}$. We consider two kinds of NNs to serve as feature extractors, i.e., convolutional neural network (CNN) and bidirectional long short term memory (BiLSTM). We pick these two because the effectiveness of both CNN [3], [4] and BiLSTM [38], [39] have been demonstrated in prior works on processing Stack Overflow textual data. Thus, we investigate these two types of NN models in our experiment. Noted that they can be replaced by other NN models and we leave it for future work. To answer RQ2, we compare the performance of a pair of variants that are only different on used NN models, i.e., Post2Vec$_{CNN,All,Sep}$ and Post2Vec$_{LSTM,All,Sep}$. Similarly, to answer RQ3 and RQ4, we implement and compare pairs of variants of Post2Vec that are only different in terms of considered components (with or without code snippets, i.e., Post2Vec$_{CNN,All,Sep}$ and Post2Vec$_{CNN,-Code,Sep}$) and architecture (single NN model or multiple NN models with different components, i.e., Post2Vec$_{CNN,All,Com}$ and Post2Vec$_{CNN,All,Sep}$).

### 3.2 Dataset

In this work, we use a snapshot from the Stack Overflow data dump following prior studies [40], [41], [42]. Zhou *et al.* [3] also use this dataset in the evaluation of their deep learning solutions. However, after careful investigation, we find that the dataset used by Zhou *et al.* has some limitations:

(1) Zhou *et al.* only used the posts before July 1, 2014. However, this dataset is too old (more than four years ago) to demonstrate the effectiveness of tag recommendation approaches on the recent posts, especially due to the rapid growth of Stack Overflow in recent years.
(2) In the experiments performed by Zhou *et al.*, posts in test data were randomly selected from the whole Stack Overflow dataset without consideration of when the posts were made. In other words, some of the newer posts were used to train the model while some older posts were used in the test dataset which introduced bias for evaluation.
(3) While millions of posts were considered by Zhou *et al.* in their experiments, only 10,000 (i.e., less than 0.1 percent) of them were selected to form the test set. Thus, the proportion of test data may not be representative enough to evaluate the approaches properly.

To address these limitations, we construct a more comprehensive dataset in this work. To address the first limitation, we use the version of Stack Overflow data dump released on September 5, 2018.[5] A commonly used first data preprocessing step in tag recommendation studies [1], [2], [3] is to identify the rare tags. A tag is *rare* if it appears less or equal to a predefined threshold *ts*. Rare tags are less important and less useful to serve as *representative tags* to be recommended to users [1]. Following the same setting as Zhou *et al.*, we set the value to the threshold *ts* as 50. Using this setting, we identify 29,357 rare tags. We also obtain 10,379,014 posts after removing the posts that only contain rare tags. To address the second limitation, we sort all the posts by the attribute *creation date*. To avoid the bias, we use the latest posts as test data and the earlier posts as training data. Lastly, to address the third limitation, the latest 100,000 posts are selected as test data which is 20 times larger than the size of test data considered by Zhou *et al.* In this way, we build a new dataset; it has 10,279,014 posts as training data, and 100,000 posts as test data. Moreover, in the preprocessing step mentioned in Section 2.2, we also construct a vocabulary for each component. They are, $\mathcal{V}^{\mathrm{T}}$ (title's vocabulary), $\mathcal{V}^{\mathrm{D}}$ (description's vocabulary), and $\mathcal{V}^{\mathrm{C}}$ (code snippets' vocabulary) that include 28,213, 119,190, and 502,466 unique tokens respectively.

### 3.3 NNs and Hyperparameter Setting

We employ independent NN models (i.e., $\mathrm{NN}_{title}$, $\mathrm{NN}_{description}$, and $\mathrm{NN}_{code}$ in Fig. 2) to extract features from the title, description, and code snippets. In our experiment, we investigate two types of NN models, i.e., CNN and BiLSTM.

4. https://pan.baidu.com/s/1pKCpodP.
5. https://archive.org/download/stackexchange.

For a fair comparison, both CNN and BiLSTM represent posts in the same dimension. Moreover, we use Adam as the optimization algorithm [26] and shuffled mini-batches to train them [43]. Inspired by [44], we investigate the number of epochs from 10 to 20. And we observe that the performance becomes stable after the number of epochs equals 16. Hence, we fix the number of epochs to 16. The parameter setting is widely used in many deep learning based approaches for software engineering tasks, e.g., [45], [46], [47]. Next, we describe the specific hyperparameter settings for both CNN- and BiLSTM-based Post2Vec.

*Convolutional Neural Network (CNN) based Post2Vec.* Each CNN model includes a convolutional layer with multiple filters and a non-linear activation function (i.e., RELU). Each filter, followed by a RELU activation function, is employed to a window of $k$ tokens to extract the corresponding features of this window. We apply a max pooling operation [48] to obtain the highest feature value of this window. Post2Vec uses multiple filters (with *K* different window sizes) to obtain features from each component. Following prior works (e.g., [45]), we follow a parameter tuning step to estimate good values of these parameters. We set aside 10 percent of the training data as a validation set (10 percent), while the remainder of the data is used by Post2Vec to learn post representation. We use the validation set for the parameter tuning procedure. Specifically, we vary different values of the parameters and measure the corresponding performance on the validation set. Take the learning rate as an example; we try setting it as 0.01, 0.001, or 0.0001. And then, we measure the performance of our models with different learning rates and find that Post2Vec achieves the best performance when the training rate is set to 0.0001. Similarly, we identify the optimal values of some other parameters. As a result of this parameter tuning step, we set the learning rate as 0.0001, the batch size as 128, and the number of filters as 100 for each of the CNN models [31]. We set window sizes (*K*) as 1, 2, 3 for title and description and 2, 3, 4 for code snippets (see Section 2.4). In this way, Post2Vec represents each component (i.e., title, description, or code snippets) of a post as a 300-dimensional ($3 \times 100$) vector, while each post is represented as a 900-dimensional vector.

*Bidirectional Long Short Term Memory (BiLSTM) based Post2Vec.* Each BiLSTM model includes a recurrent layer with a hidden state. We set the number of features in the hidden state to 300. As a result, each component is represented as a 300-dimensional vector while each post is represented as a 900-dimensional vector.

### 3.4 Evaluation Metrics

To evaluate the performance of their proposed approaches, Zhou *et al.* [3] used Precision@k, Recall@k, and F1-score@k. The 3 metrics are defined as the averages of Precision@$k_i$, Recall@$k_i$, and F1-score@$k_i$ for each post $p_i$ respectively. The latter metrics are defined below:

*Precision@$k_i$* is the percentage of a post's ground truth tags $GT_i$ that are in the top-k recommended list $TR_i^k$, formally defined as:

$$\text{Precision@}k_i = \frac{|TR_i^k \cap GT_i|}{k}. \tag{5}$$

*Recall@$k_i$* is the percentage of tags in the top-k recommended list $TR_i^k$ that are among the set of ground truth tags $GT_i$. However, Recall@$k_i$ value disfavors small $k$ [3], [37]. For example, for a post $p_i$ with 2 ground truth tags, even if the first 2 recommended tags are correct, the Recall@$1_i$ value is still 50 percent (i.e., 1/2). Hence, we follow the definition of Recall@$k_i$ that was used to evaluate the state-of-the-art tag recommendation approach by Zhou *et al.* [3]. In Zhou *et al.*'s work, if the number of ground truth tags of a post is larger than $k$, the value of Recall@$k_i$ is computed using the same formula as Precision@$k_i$. In this way, for the aforementioned example, Recall@$1_i$ will be 100 percent. This perfect score better reflects the correctness of the recommended tags at top-1 position (as it is the best result possible). Zhou *et al.*'s Recall@$k_i$ is formally defined as:

$$\text{Recall@}k_i = \begin{cases} \frac{|TR_i^k \cap GT_i|}{k}, |GT_i| > k \\ \frac{|TR_i^k \cap GT_i|}{|GT_i|}, |GT_i| \leq k \end{cases}. \tag{6}$$

*F1-score@$k_i$* is a harmonic mean of *Precision@$k_i$* and *Recall@$k_i$*, and it is formally defined as:

$$\text{F1} - \text{score@}k_i = 2 \times \frac{\text{Precision@}k_i \times \text{Recall@}k_i}{\text{Precision@}k_i + \text{Recall@}k_i}. \tag{7}$$

Note that, based on the definition of Precision@k (Equation (5)), Recall@k (Equation (6)), and F1-score@k (Equation (7)), their values will always be the same when $k$ equals 1; this is the case because each post is labeled by at least one tag, and thus $|GT_i| = k$, when $k = 1$. Moreover, the value of Zhou *et al.*'s Recall@$k_i$ formula may not always increase with an increase in the value of $k$ because of the second subformula in Equation (6).

Zhou *et al.* [3] used Precision@k, Recall@k, and F1-score@k, where k $\in \{5, 10\}$. However, we find that Stack Overflow only allows users to assign at most 5 tags for each post. Thus, in this work, we use more proper settings of k, i.e., k $\in \{1, 2, 3, 4, 5\}$. Similar to Zhou *et al.*'s work, we regard F1-score@k as the main evaluation metric as it takes into consideration both Precision@k and Recall@k.[6]

### 3.5 Experimental Result

*Effectiveness Comparison.* Table 2 shows a comparison in terms of effectiveness. The results show that Post2Vec$_{CNN,All,Sep}$ achieves the best performance among all variants and baselines on all evaluation metrics. Post2Vec$_{LSTM,All,Sep}$ achieves the second best performance, while TagRCNN performs the worst. Specifically, Post2Vec$_{CNN,All,Sep}$ outperforms Post2Vec$_{LSTM,All,Sep}$, Post2Vec$_{CNN,-Code,Sep}$, Post2Vec$_{CNN,All,Com}$, TagCNN, and TagRCNN by 2, 7, 5, 15, and 25 percent in terms of F1-score@5, respectively.

*Efficiency Comparison.* For a fair comparison, all the approaches were run on the same machine: a 64-bit, 14.04.4

6. In this work, by default, we use the evaluation metrics that were used to evaluate the state-of-the-art approach [3]. We have also considered other evaluation metrics. They are described in the Appendix A that we include as a supplementary material, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/TSE.2021.3093761.

TABLE 2
Effectiveness of Post2Vec and Its Baselines

| | Precision@1 | Precision@2 | Precision@3 | Precision@4 | Precision@5 |
|---|---|---|---|---|---|
| TagRCNN | 0.68 | 0.53 | 0.42 | 0.34 | 0.29 |
| TagCNN | 0.70 | 0.56 | 0.45 | 0.37 | 0.32 |
| Post2Vec$_{LSTM,All,Sep}$ | 0.78 | 0.62 | 0.50 | 0.41 | 0.35 |
| Post2Vec$_{CNN,-Code,Sep}$ | 0.74 | 0.59 | 0.48 | 0.40 | 0.34 |
| Post2Vec$_{CNN,All,Com}$ | 0.77 | 0.62 | 0.50 | 0.41 | 0.35 |
| Post2Vec$_{CNN,All,Sep}$ | 0.79 | 0.63 | 0.51 | 0.42 | 0.36 |
| | Recall@1 | Recall@2 | Recall@3 | Recall@4 | Recall@5 |
| TagRCNN | 0.68 | 0.57 | 0.53 | 0.52 | 0.54 |
| TagCNN | 0.70 | 0.60 | 0.56 | 0.56 | 0.58 |
| Post2Vec$_{LSTM,All,Sep}$ | 0.78 | 0.67 | 0.62 | 0.63 | 0.65 |
| Post2Vec$_{CNN,-Code,Sep}$ | 0.74 | 0.64 | 0.60 | 0.61 | 0.63 |
| Post2Vec$_{CNN,All,Com}$ | 0.77 | 0.66 | 0.62 | 0.63 | 0.65 |
| Post2Vec$_{CNN,All,Sep}$ | 0.79 | 0.68 | 0.64 | 0.64 | 0.66 |
| | F1-score@1 | F1-score@2 | F1-score@3 | F1-score@4 | F1-score@5 |
| TagRCNN | 0.68 | 0.54 | 0.46 | 0.40 | 0.36 |
| TagCNN | 0.70 | 0.57 | 0.49 | 0.43 | 0.39 |
| Post2Vec$_{LSTM,All,Sep}$ | 0.78 | 0.63 | 0.54 | 0.48 | 0.44 |
| Post2Vec$_{CNN,-Code,Sep}$ | 0.74 | 0.61 | 0.52 | 0.46 | 0.42 |
| Post2Vec$_{CNN,All,Com}$ | 0.77 | 0.63 | 0.54 | 0.48 | 0.43 |
| Post2Vec$_{CNN,All,Sep}$ | 0.79 | 0.64 | 0.55 | 0.49 | 0.45 |

LTS GPU server with Tesla P100-SXM2-16GB.[7] Table 3 shows the comparison in terms of training time cost; we observed that all approaches cost more than 5 days to train the model due to the large dataset. Specifically, TagRCNN requires 8.5 days, TagCNN requires 7.1 days, Post2Vec$_{CNN,All,Sep}$ requires 7 days, Post2Vec$_{LSTM,All,Sep}$ requires 12.7 days, Post2Vec$_{CNN,-Code,Sep}$ requires 6.9 days, and Post2Vec$_{CNN,All,Com}$ requires 5.4 days. Although the training time takes days, the time these approaches take to recommend a set of tags to a post (i.e., model application time) is very fast. It took each approach 0.08 seconds per post on average. Note that training can be done once in a long while.

### 3.6 Research Questions and Analysis

We perform further analysis to answer the three research questions.

*RQ1: Compared with the state-of-the-art approach, how effective and efficient is Post2Vec in tag recommendation?*

To answer RQ1, we compare Post2Vec with the state-of-the-art tag recommendation approaches (i.e., TagCNN and TagRCNN). In terms of effectiveness, our experimental results show that Post2Vec largely outperforms the baseline approaches, i.e., it improves over TagCNN and TagRCNN in terms of F1-score@5 by 15 and 25 percent, respectively. Moreover, in terms of efficiency, Post2Vec costs 0.1 day less than TagCNN and 1.5 days less than TagRCNN in terms of model training time. In terms of model application time (i.e., the time it takes to recommend tags to a new SO post), all approaches take a fraction of a second.

> *Post2Vec outperforms the state-of-the-art approaches in terms of both effectiveness (by 15-25 percent in terms of F1-score@5) and efficiency (by up to 1.5 days in terms of model training time).*

*RQ2: What is the impact of different types of NNs as feature extractors?*

To answer RQ2, we compare the performance of a pair of Post2Vec's variants that only differ in the way of used NN model, i.e., Post2Vec$_{CNN,All,Sep}$ and Post2Vec$_{LSTM,All,Sep}$. From Table 2, we find that Post2Vec$_{CNN,All,Sep}$ outperforms Post2Vec$_{LSTM,All,Sep}$ by 3 percent in term of F1-score@5. We apply the Wilcoxon Signed Rank Test [49] at a 95 percent confidence level (i.e., p-value $<$ 0.05) on the paired data which corresponds to the F1-score@5 of CNN- and LSTM-based Post2Vec. We find that CNN-based Post2Vec (i.e., Post2Vec$_{CNN,All,Sep}$) significantly outperforms LSTM-based Post2Vec (i.e., Post2Vec$_{LSTM,All,Sep}$). According to Table 3, we also find that Post2Vec$_{LSTM,All,Sep}$ costs 5.6 more days than Post2Vec$_{CNN,All,Sep}$ for model training.

> *CNN-based Post2Vec outperforms LSTM-based Post2Vec significantly and consumes fewer computing resources during model training.*

*RQ3: Does Post2Vec benefit from code snippets?*

To answer RQ3, we compare the performance of Post2Vec$_{CNN,All,Sep}$ and Post2Vec$_{CNN,-Code,Sep}$ since the only difference between them is the consideration of code snippets. The experimental results show that Post2Vec$_{CNN,All,Sep}$ outperforms Post2Vec$_{CNN,-Code,Sep}$ in terms of effectiveness (7 percent better) but requires more time to train (0.1 day

7. https://www.nvidia.com/en-us/data-center/tesla-p100.

TABLE 3
Training Time Cost Comparison

| Approach | Time cost (Mins) |
|---|---|
| TagRCNN | 12,287 |
| TagCNN | 10,278 |
| Post2Vec$_{LSTM,All,Sep}$ | 18,233 |
| Post2Vec$_{CNN,-Code,Sep}$ | 9,916 |
| Post2Vec$_{CNN,All,Com}$ | 7,811 |
| Post2Vec$_{CNN,All,Sep}$ | 10,130 |

more). It is reasonable that Post2Vec$_{CNN,-Code,Sep}$ is faster to train than Post2Vec$_{CNN,All,Sep}$ since Post2Vec$_{CNN,-Code,Sep}$ has a similar but simpler architecture (i.e., it ignores the code snippets).

*Consideration of code snippets can boost effectiveness (i.e., a boost in F1-score@5 by 7 percent) but consumes more computing resources during model training.*

*RQ4: What is the impact of handling post components separately rather than combining them together?*

To answer RQ4, we compare the performance of Post2Vec$_{CNN,All,Com}$ and Post2Vec$_{CNN,All,Sep}$ because both of them utilized title, description, and code snippets as input data but Post2Vec$_{CNN,All,Sep}$ separately employs three CNN models for the title while description, and code snippets, Post2Vec$_{CNN,All,Com}$ concatenates them as one document and feed it to a single CNN model. The experimental results show that Post2Vec$_{CNN,All,Sep}$ outperforms Post2Vec$_{CNN,All,Com}$ over all the evaluation metrics; in terms of F1-score@5, Post2Vec$_{CNN,All,Com}$ is 5 percent better than Post2Vec$_{CNN,All,Com}$. We apply the Wilcoxon Signed Rank Test [49] at a 95 percent confidence level (i.e., p-value $< 0.05$) on the paired data which corresponds to the F1-score@5 and we find that Post2Vec$_{CNN,All,Sep}$ significantly outperforms Post2Vec$_{CNN,All,Com}$. But Post2Vec$_{CNN,All,Com}$ is 22 percent more efficient.

*Post2Vec's architecture that considers different post components (i.e., title, description, and code snippets) separately significantly boosts the performance by 5 percent in terms of F1-score@5 but consumes more computing resources than considers different components together during model training.*

For the rest of the paper, we refer the variant Post2Vec$_{CNN,All,Sep}$ to Post2Vec.

## 4 Post2Vec for Downstream Tasks

As stated earlier, the goal of this work is to build a Stack Overflow post representation that can be useful for multiple downstream tasks. In the training phase, our approach uses tag as a guide to aid the learning of post representation. A tag is a word or phrase that is attached to a post as a result of a crowd sourced process and serves as a semantic label of the post. Downstream tasks that are poised to benefit more by representations learned by our approach are those where: (1) the downstream task can benefit from an abstraction of the post; (2) the tags in our training data capture suitable semantics that are needed for the task. In this section, we pick three tasks that satisfy the above two criteria.

### 4.1 Baseline Approaches

We first compare against state-of-the-art approaches proposed by Xu *et al.* [6], Beyer *et al.* [10], and Huang *et al.* [17] for relatedness prediction, post classification, and API recommendation tasks, respectively. Note that all the state-of-the-art approaches do not involve deep learning. Furthermore, we also compare the performance of our approach with two deep learning based techniques, i.e., TagCNN [3][8] and Doc2Vec [16], that can also be used to generate post representations.

TagCNN forms a single output vector (i.e., post vector) in the penultimate layer, and subsequently pass the vector to a fully connected softmax layer to compute the probability distribution over tags. Thus, we fed the posts used in the downstream tasks to the TagCNN model which was trained for the tag recommendation task (refer to Section 3) and collected the corresponding vectors of those posts. Moreover, as posts are also documents, we use Doc2Vec (i.e., a document embedding algorithm) as another baseline approach. We follow the Doc2Vec setting used in [15] as they also use Doc2Vec to analyze Stack Overflow posts. We use Gensim [50] to implement the Doc2Vec model. For a fair comparison, the same training dataset is used to build Post2Vec, TagCNN, and Doc2Vec models, i.e., the training dataset used for the tag recommendation task.

### 4.2 Task 1: Relatedness Prediction

A post on a software question and answer (SQA) site can be considered as a *knowledge unit*. Often information in one post is insufficient to address a developer's information need. For such cases, developers need to search for related posts that provide additional relevant knowledge units to solve their technical problems. The task of automatically identifying such related posts or knowledge units is described as *relatedness prediction*. It can support developers for different purposes; for example, searching for related solutions to better solve a particular problem, or gathering new knowledge by browsing related SQA posts [4], [6].

#### 4.2.1 Problem Formulation

The *relatedness prediction* is formulated as a multi-class classification problem. Given a set of pairs of software posts, labels are assigned to them to describe how related they are. The labels are based on the order of relatedness from most to least related, i.e., *Duplicate* > *Direct* > *Indirect* > *Isolated*. There is only one possible relatedness label between two posts. For more details, please refer to the original work [4].

#### 4.2.2 State-of-the-art Approach

The state-of-the-art *relatedness prediction* approach named SoftSVM was proposed by Xu *et al.* [6]. SoftSVM introduced

8. We do not include tag recommendation approaches that do not use deep learning as baselines for downstream tasks since they cannot be used to generate post representations.

a soft-cosine similarity which is able to capture the degree of relatedness when the text of posts share related words. To measure the degree of relatedness, SoftSVM constructs a four-dimensional vector which corresponds to four types of features, 1) cosine similarity based on Stack Overflow data $cos_{SO}$, 2) soft-cosine similarity based on Stack Overflow data $soft_{SO}$, 3) soft-cosine similarity based on pre-trained Google word2vec $soft_{Google}$, and 4) soft-cosine similarity based on Levenshtein distance $soft_{Edit}$. For more details on these four features, please refer to [6]. To summarize, for a given pair of posts, a four-dimensional feature vector $FV_{SoftSVM}$ is constructed where each dimension corresponds to each feature value, i.e., $\langle cos_{SO}, soft_{SO}, soft_{Google}, soft_{Edit}\rangle$.

#### 4.2.3 Dataset

We use the same dataset used in the state-of-the-art work [6]. The dataset contains 40,000 pairs of posts with the four types of relatedness. Detailedly, each type of relatedness corresponds to 10,000 pairs of posts, i.e., the dataset is balanced with respect to class.

#### 4.2.4 Experimental Setting

We hypothesize that there is a correlation between the post vector distance and the relatedness, i.e., the smaller the distance between two generated post vectors, the more related these two posts are. To evaluate the post representation generated by the three approaches (i.e., Post2Vec, TagCNN and Doc2Vec), we consider post vector distance as an extra feature and combine it with the features used in the state-of-the-art relatedness prediction approach. In particular, we add the euclidean distance [51] (see Equation (8)) between two post vectors as an additional feature, namely $Dist$. Then, we append $Dist$ to the feature vector used by the state-of-the-art approach. In this way, we construct a five-dimensional feature vector $FV_{SoftSVM+Dist}$ for each pair of posts, i.e., $\langle cos_{SO}, soft_{SO}, soft_{Google}, soft_{Edit}, Dist\rangle$. We name the feature vector based on Post2Vec, TagCNN, Doc2Vec as $FV_{SoftSVM+P2V}$, $FV_{SoftSVM+TagCNN}$, and $FV_{SoftSVM+D2V}$, respectively. To facilitate comparison, we feed the feature vector $FV_{SoftSVM+Dist}$ to the same kind of traditional machine learning classifier (i.e., SVM) with the same setting as described in [6]. Implementation-wise, we reuse the source code released by Xu *et al.*[9] and modify the feature vector part. Then, ten times ten-fold cross-validation is performed to evaluate predictive models.

$$Dist(\mathbf{p}, \mathbf{q}) = Dist(\mathbf{q}, \mathbf{p}) = \sqrt{\sum_{i=1}^{Dimension} (q_i - p_i)^2}. \quad (8)$$

#### 4.2.5 Evaluation Metrics

We use precision, recall, and F1-score, which were also used by Xu *et al.* as evaluation metrics. The definitions of precision, recall, and F1-score are as below. Note that $C_{ij}$ denotes the number of pairs of class $i$ that are predicted as of class $j$. In the following equations, $K = 4$, as there are 4 relatedness types: *Duplicate*, *Direct*, *Indirect* and *Isolated*).

*Precision* for a class $i$ is the percentage of post pairs correctly classified as the class $i$ among all pairs predicted as the class $i$.

$$\text{Precision}_i = \frac{C_{ii}}{\sum_{1 \le j \le K} C_{ji}}. \quad (9)$$

*Recall* for a class $i$ is the percentage of post pairs correctly classified as the class $i$ among all pairs whose ground truth label is class $i$.

$$\text{Recall}_i = \frac{C_{ii}}{\sum_{1 \le j \le K} C_{ij}}. \quad (10)$$

*F1-score* for a class $i$ is a harmonic mean of precision and recall for that class.

$$\text{F1} - \text{score}_i = 2 \times \frac{\text{Precision}_i \times \text{Recall}_i}{\text{Precision}_i + \text{Recall}_i}. \quad (11)$$

Higher values of the above metrics indicate better performance. Following [6], F1-score is regarded as the main metric.

#### 4.2.6 Experimental Result

Table 4 shows the performance obtained using the feature vector $FV_{SoftSVM}$ and $FV_{SoftSVM+Dist}$ generated by the three approaches. The results of the experiment show that $FV_{SoftSVM+P2V}$ achieves the best performance, $FV_{SoftSVM+TagCNN}$ achieves the second best, and $FV_{SoftSVM}$ achieves worst in terms of precision, recall, and F1-score. We found that when using $FV_{SoftSVM+P2V}$ as feature vector, in terms of overall F1-score, it outperforms $FV_{SoftSVM+TagCNN}$, $FV_{SoftSVM+D2V}$, and $FV_{SoftSVM}$ by 2, 8, and 10 percent, respectively. We apply the Wilcoxon Signed Rank Test [49] at a 95 percent significance level (i.e., p-value $< 0.05$) on the paired data which corresponds to the F1-score of our approach, i.e., Post2Vec, and the best performing baseline approach, i.e., TagCNN. We find that Post2Vec significantly outperforms TagCNN in terms of F1-score.

Overall, we are able to boost the performance of the state-of-the-art relatedness prediction approach by leveraging the post vectors generated by Post2Vec.

### 4.3 Task 2: Post Classification

Beyer *et al.* [10] presented a taxonomy for Android-related Stack Overflow (SO) posts and manually labeled 500 posts. The taxonomy includes seven categories in which an Android post can be classified, namely, API CHANGE, API USAGE, CONCEPTUAL, DISCREPANCY, LEARNING, ERRORS, and REVIEW.

#### 4.3.1 Problem Formulation

The post classification problem is formulated as a multi-label classification problem. Given a set of software question and answer posts that are already labeled with one or multiple categories, a new post needs to be classified into a set of appropriate categories (i.e., API CHANGE, API USAGE, CONCEPTUAL, DISCREPANCY, LEARNING, ERRORS, and REVIEW).

9. https://github.com/XBWer/ESEM2018.

TABLE 4
Performance Comparison for Relatedness Prediction

| | Feature Vector | Duplicate | Direct | Indirect | Isolated | Overall |
|---|---|---|---|---|---|---|
| Precision | $FV_{SoftSVM}$ | 0.47 | 0.41 | 0.43 | 0.73 | 0.51 |
| | $FV_{SoftSVM+D2V}$ | 0.48 | 0.41 | 0.44 | 0.73 | 0.51 |
| | $FV_{SoftSVM+TagCNN}$ | 0.53 | 0.44 | 0.46 | 0.76 | 0.55 |
| | $FV_{SoftSVM+P2V}$ | 0.53 | 0.45 | 0.48 | 0.77 | **0.56** |
| Recall | $FV_{SoftSVM}$ | 0.54 | 0.19 | 0.49 | 0.91 | 0.53 |
| | $FV_{SoftSVM+D2V}$ | 0.50 | 0.23 | 0.50 | 0.91 | 0.54 |
| | $FV_{SoftSVM+TagCNN}$ | 0.52 | 0.33 | 0.49 | 0.91 | 0.56 |
| | $FV_{SoftSVM+P2V}$ | 0.61 | 0.31 | 0.48 | 0.91 | **0.58** |
| F1-score | $FV_{SoftSVM}$ | 0.51 | 0.26 | 0.46 | 0.81 | 0.51 |
| | $FV_{SoftSVM+D2V}$ | 0.49 | 0.30 | 0.47 | 0.81 | 0.52 |
| | $FV_{SoftSVM+TagCNN}$ | 0.52 | 0.38 | 0.47 | 0.83 | 0.55 |
| | $FV_{SoftSVM+P2V}$ | 0.57 | 0.36 | 0.48 | 0.83 | **0.56** |

#### 4.3.2 State-of-the-art Approach

To classify posts into multiple categories, Beyer *et al.* [10] constructed a binary classifier for each category. To construct a classifier, they considered 82 different configurations based on, e.g., whether data balancing is performed, which classifier is used, whether parts-of-speech tags are included in the feature vector representing a post, etc. The best configuration is then determined for each category, and the corresponding results are reported.

#### 4.3.3 Dataset

To facilitate comparison, we have used the same dataset used by Beyer *et al.* [10]. The dataset contains 500 posts which have been manually labeled, with 30, 206, 145, 129, 79, 30, and 93 posts labeled as API CHANGE, API USAGE, CONCEPTUAL, DISCREPANCY, LEARNING, ERRORS, and REVIEW, respectively.

#### 4.3.4 Experimental Setting

Our goal is to investigate whether leveraging post representation generated using Post2Vec, TagCNN, and Doc2Vec can boost the effectiveness of Beyer *et al.*'s state-of-the-art approach. Thus, we reuse most of Beyer *et al.*'s proposed architecture that considers the 82 configurations and determines the best one. We refer to Beyer *et al.*'s original approach as $FV_{STOA}$. The only change that we made is to enhance the feature vector that $FV_{STOA}$ uses to represent a post. Specifically, we construct a new feature vector for each post by appending $FV_{STOA}$'s feature vector with the corresponding post's representation that is learned by using either Post2Vec, TagCNN or Doc2Vec. We refer to the approaches that use the feature vectors enhanced with post representation learned using Post2Vec, TagCNN or Doc2Vec as $FV_{STOA+P2V}$, $FV_{STOA+TagCNN}$, and $FV_{STOA+D2V}$ respectively. Implementation-wise, we reused the source code released by Beyer *et al.*[10] and only modified the feature vector part.

Beyer *et al.* divided their manually-labeled set of 500 posts into training and test sets, with 90 percent (i.e., 450) of the posts used for training and 10 percent (i.e., 50) for testing. Since the dataset is unbalanced, random stratified sampling [52] is applied to ensure that the test set contains 10 percent or at least three posts of each category. The experiment is repeated 50 times using the random stratified sampling, and the average scores of the evaluation metrics (described below) are reported.

#### 4.3.5 Evaluation Metrics

We use precision, recall, and F1-score, which were also used as evaluation metrics by Beyer *et al.* For each category, these metrics are defined for both sides of the classification: whether a post is classified correctly as belonging to the category ($class_T$) and whether a post is classified correctly as not belonging to the category ($class_F$). For all of the above metrics, higher values indicate better performance. F1-score for $class_T$ (denoted as $f_T$) is regarded as the main metric since the dataset is unbalanced and the minority is $class_T$ (i.e., belonging to a category).

#### 4.3.6 Experimental Result

Table 5 shows the performance of the post classification task focusing on $f_T$ and the other two metrics used to derive it ($pre_T$ and $rec_T$). The experimental results show that $FV_{STOA+P2V}$ achieves the best average number of $f_T$ and also the best performance for the most number of categories. Specifically, across the 7 categories, on average $FV_{STOA+P2V}$ outperforms $FV_{STOA}$ by 7 percent in terms of $f_T$. We apply the Wilcoxon Signed Rank Test [49] at a 95 percent significance level (i.e., p-value $<$ 0.05) on the paired data which corresponds to the $f_T$ of the feature vector based on our approach, i.e., $FV_{STOA+P2V}$, and the best performing baseline approach, i.e., $FV_{STOA}$. We find that $FV_{STOA+P2V}$ significantly outperforms $FV_{STOA}$ in terms of $f_T$. Moreover, we find that $FV_{STOA+D2V}$ and $FV_{STOA+TagCNN}$ perform worse than $FV_{STOA}$. Also, none of them achieve the best performance for a single category. It indicates that Post2Vec's learned representation enhances the original feature vector, while the same cannot be said for representations learned by TagCNN and Doc2Vec.

10. https://github.com/icpc18submission34/icpc18submission34.

TABLE 5
Performance Comparison for Post Classification

| Category | $pre_T$ | | | |
|---|---|---|---|---|
| | $FV_{STOA}$ | $FV_{STOA+D2V}$ | $FV_{STOA+TagCNN}$ | $FV_{STOA+P2V}$ |
| API_CHANGE | 0.59 | 0.60 | 0.74 | 0.67 |
| API_USAGE | 0.91 | 0.82 | 0.94 | 0.88 |
| CONCEPTUAL | 0.59 | 0.47 | 0.62 | 0.58 |
| DISCREPANCY | 0.41 | 0.46 | 0.48 | 0.53 |
| DOCUMENTATION | 0.56 | 0.35 | 0.40 | 0.64 |
| ERRORS | 0.88 | 0.86 | 0.56 | 0.93 |
| REVIEW | 0.42 | 0.38 | 0.47 | 0.53 |
| **Average** | 0.62 | 0.56 | 0.60 | **0.68** |
| | $rec_T$ | | | |
| | $FV_{STOA}$ | $FV_{STOA+D2V}$ | $FV_{STOA+TagCNN}$ | $FV_{STOA+P2V}$ |
| API_CHANGE | 0.95 | 0.76 | 0.73 | 0.77 |
| API_USAGE | 0.78 | 0.58 | 0.58 | 0.71 |
| CONCEPTUAL | 0.62 | 0.74 | 0.46 | 0.68 |
| DISCREPANCY | 0.87 | 0.63 | 0.69 | 0.76 |
| DOCUMENTATION | 0.55 | 0.27 | 0.18 | 0.38 |
| ERRORS | 0.33 | 0.21 | 0.34 | 0.70 |
| REVIEW | 0.54 | 0.49 | 0.25 | 0.48 |
| **Average** | 0.66 | 0.53 | 0.46 | 0.64 |
| | $f_T$ | | | |
| | $FV_{STOA}$ | $FV_{STOA+D2V}$ | $FV_{STOA+TagCNN}$ | $FV_{STOA+P2V}$ |
| API_CHANGE | 0.71 | 0.65 | 0.71 | 0.68 |
| API_USAGE | 0.84 | 0.68 | 0.71 | 0.78 |
| CONCEPTUAL | 0.59 | 0.57 | 0.52 | 0.62 |
| DISCREPANCY | 0.56 | 0.53 | 0.56 | 0.62 |
| DOCUMENTATION | 0.53 | 0.28 | 0.24 | 0.45 |
| ERRORS | 0.46 | 0.33 | 0.37 | 0.79 |
| REVIEW | 0.46 | 0.42 | 0.31 | 0.48 |
| **Average** | 0.59 | 0.49 | 0.49 | **0.63** |

*$pre_T$, $rec_T$, $f_T$ = precision, recall and F1-score for $class_T$.*

## 4.4 Task 3: API Recommendation

According to a survey conducted by Huang *et al.* [17], API recommendation approach is desired and Stack Overflow is an important resource for developers to search APIs.

### 4.4.1 Problem Formulation

In the literature, the API recommendation problem is formed as given a query in natural language that describes a technical question, recommending a certain API or API sequence to answer the question [17], [53], [54]. Depending on the granularity of the recommended API, the problem is divided into two categories, method- and class-level API recommendation.

### 4.4.2 State-of-the-Art Approach

Huang *et al.* [17] focus on the API recommendation at the method level and present an approach named BIKER which recommends API based on Stack Overflow data. The framework of BIKER mainly contains three stages. First, given a query and Stack Overflow data dump, BIKER collects the top *k* most similar questions for the query by measuring the similarity between the questions and the given query. Second, it extracts API from the answers of the *k* similar questions and ranks them by computing similarity between each API candidate and the query. Third, BIKER selects the top 5 most similar APIs and summarizes them by considering multiple sources, e.g., the information in the official description, title of the similar questions, and the code snippets exist in the answers of the similar questions.

### 4.4.3 Experimental Setting

In the first stage of BIKER, it computes the similarity between each question in the Stack Overflow data dump and the query based on a word IDF (inverse document frequency) vocabulary and a word embedding model. Based on the similarity, all the questions are ranked and top *k* of them are selected as similar questions to the query.

To investigate whether leveraging post representation generated using Post2Vec, TagCNN, and Doc2Vec can boost the effectiveness of BIKER, we embed the post representation into the rank list of the similar questions based on a new perspective, i.e., considering similarity between questions based on the post representation. Given a query, we first collect the rank list with top *k* most similar questions and their similarities to the query returned by BIKER, i.e.,

$$\begin{aligned} RankList = \{&TQ_1 : \{Sim\langle TQ_1, Query\rangle\},\\ &TQ_2 : \{Sim\langle TQ_2, Query\rangle\},\\ &\ldots,\\ &TQ_k : \{Sim\langle TQ_k, Query\rangle\}\}. \end{aligned}$$

And then we collect 5 questions in the data dump with the least euclidean distance to each of the similar questions $TQ_i$ based on the post representation, i.e.,

$$\begin{aligned} TQ_i \Rightarrow TQSimQs_i = \{&TQSimQ_i^1,\\ &TQSimQ_i^2,\\ &\ldots,\\ &TQSimQ_i^5\} \end{aligned}$$

We set the similarity between two questions as $Sim\langle Q_i, Q_j\rangle = 1/(1 + dist\langle Q_i, Q_j\rangle)$. Last, we add the top 5 questions (i.e., $TQSimQs$) to the rank list $RankList$. For each question $TQSimQ_i^j$, we set its similarity to the query by multiplying two similarities, (1) similarity between the corresponding top question and query, (2) similarity between the corresponding top question and the question, i.e.,

$$\begin{aligned} Sim\langle TQSimQ_i^j, Query\rangle = &Sim\langle Q_i, Query\rangle\\ &\times Sim\langle TQ_i, TQSimQ_i^j\rangle \end{aligned}$$

In this way, the rank list of similar questions is expanded and its top *k* questions are fed to the next stage of BIKER. We follow the same setting in [17] and set *k* as 50.

### 4.4.4 User Study

To investigate whether BIKER can help developers find the appropriate APIs more efficiently and accurately in practice, Huang *et al.* conducted a user study [17]. Thus, we conduct a user study to evaluate the performance of two versions of BIKER, i.e., before and after integrating the post representation.

*Experimental queries and ground-truth APIs*. In [17], 10 queries are selected as experimental queries. In this work, we randomly select 50 queries which are 5 times more. And for each query, we collect its corresponding API or API sequence which is regarded as the ground truth.

*Participants*. We recruited 8 participants from the first author's university, 1 postdoc, 3 PhDs, and 4 research engineers. The years of their developing experience vary from 3 to 7 years, with an average of 4.6 years.

*Experimental Groups*. We divided the participants uniformly based on years of development experience into four groups, (1) *BIKER*: the full-feature of original BIKER, (2) *BIKER+D2V*: the enhanced BIKER which integrated the post representation learned by doc2vec, (3) *BIKER*

TABLE 6
Performance Comparison for API Recommendation

| | Correctness | Time Cost (in Seconds) |
|---|---|---|
| BIKER | 0.40 | 67 |
| BIKER+D2V | 0.39 | 80 |
| BIKER+TagCNN | 0.37 | 108 |
| BIKER+P2V | **0.44** | **55** |

+*TagCNN*: the enhanced BIKER which integrated the post representation learned by TagCNN, (4) *BIKER+Post2Vec*: the enhanced BIKER which integrated the post representation learned by Post2Vec.

*Evaluation Metrics*. To measure the accuracy of BIKER, two evaluation metrics have been used in [17], correctness and time cost. Correctness is used to evaluate whether a participant can find the correct APIs. For a query that only needs one API method, correctness is 1 if the participant submitted only one API and it's correct, otherwise, it is 0. For a query that needs an API sequence, correctness is the proportion of the correct APIs submitted by the participant among all APIs in the correct API sequence. Time cost evaluates how fast a participant can answer a question.

*Result Analysis*. As shown in Table 6, the result shows that BIKER+Post2Vec achieves the best performance, BIKER achieves the second-best, BIKER+TagCNN performs the worst, in terms of both correctness and time cost. We find that the correctness of BIKER is improved by 10 percent and the time cost is reduced by 18 percent after integrating the post representation learned by Post2Vec. Moreover, we also find that only the representation learned by Post2Vec boosts the performance of BIKER and the other two baselines harm the performance.

Table 7 presents the recommendation result returned by BIKER and BIKER+Post2Vec for an experimental query, *Converting a time String to ISO 8601 format*. We find that the correct API (i.e., *java.time.Instant.toString()*) does not exist in all the APIs recommended by BIKER but it is ranked first by BIKER+Post2Vec. It indicates that BIKER+Post2Vec successfully identifies similar questions which cover the ground truth but BIKER fails. Thus, the result shows that integrating the post representation learned by Post2Vec can boost the performance of BIKER.

## 5 Discussion

### 5.1 Qualitative Analysis

Recall that the goal of Post2Vec is to embed posts to a vector space where it is easy to compute meaningful distances. Ideally, the distance measure between closely related posts should be small to capture the perception of similarity. Also recall that, for the relatedness prediction task, there are four relatedness types among pairs of posts, and the order of relatedness from most to least related is *Duplicate* $>$ *Direct* $>$ *Indirect* $>$ *Isolated*. The experimental results in Section 4.3 show that the state-of-the-art approach is improved by adding distance between post vectors as an additional feature.

To further investigate the capability of Post2Vec, we visualize the vector space of randomly selected 20 pairs of posts with different types of relatedness labels (details are shown in Table 8) and collected their corresponding post vectors. Fig. 4 presents the post vector space learned by Post2Vec, visualized in a two-dimensional space using the t-SNE dimensionality reduction technique [55]. Specifically, Figs. 4a, 4b, 4c, and 4d present vector space of Post2Vec for pairs of post with different types of relatedness labels, *duplicate*, *direct*, *indirect*, and *isolated*, respectively. From the figures, we observe that the rank list of average distance between pairs of posts which belong to the same type of relatedness is: *duplicate* (Fig. 4a) $<$ *direct* (Fig. 4b) $<$ *indirect* (Fig. 4c) $<$ *isolated* (Fig. 4d). In other words, post vectors are closer to each other when the corresponding pair of posts are more related. This further demonstrates the quality of post vectors generated by Post2Vec.

TABLE 7
Recommendation Returned by BIKER and BIKER+Post2Vec for an Experimental Query

| **Query:** *Converting a time String to ISO 8601 format* | | |
|---|---|---|
| **1st Recommendation** | **BIKER** | **BIKER+Post2Vec** |
| API | java.lang.String.format | **java.time.Instant.toString**✓ |
| Java Doc | Returns a formatted string using the specified format string and arguments. | A string representation of this instant using ISO-8601 representation. |
| Relevant SO Questions | 1.Convert String to UUID formatted string (25895225*)<br>2.String.format runtime formatting (19803405*)<br>3.Formatting: String cannot be converted to String[] (35702926*) | 1.Format Instant to String (25229124*) |
| Code Snippets | /**********code snippet1**********/<br>String plain = "...";<br>String uuid = String.format("...", plain.substring(0,7));<br><br>/**********code snippet2**********/<br>int width = ...;<br>String fmt = String.format("-%i%%s %%s", width);<br>String formatted = String.format(fmt, "Hello", "World");<br><br>/**********code snippet3**********/<br>dataArray +=String.format("...", b.getTitle(), b.getIsbn()); | /**********code snippet1**********/<br>DateTimeFormatter formatter = DateTimeFormatter<br>.ofPattern("...");<br>String text = date.toString(formatter);<br>LocalDate date = LocalDate.parse(text, formatter); |

TABLE 8
Pairs in Figs. 4a, 4b, 4c, and 4d

| pairId | id1 | id2 | label |
|---|---|---|---|
| 0 | 10499846 | 10499780 | duplicate |
| 1 | 41995162 | 41985606 | duplicate |
| 2 | 25265734 | 25269382 | duplicate |
| 3 | 12399031 | 12399168 | duplicate |
| 4 | 21347995 | 21579452 | duplicate |
| 0 | 9701688 | 9690628 | direct |
| 1 | 4807299 | 27781975 | direct |
| 2 | 46692249 | 43321202 | direct |
| 3 | 12884573 | 40770990 | direct |
| 4 | 16821771 | 13944155 | direct |
| 0 | 24791298 | 27667086 | indirect |
| 1 | 7183010 | 37101318 | indirect |
| 2 | 32147303 | 23247539 | indirect |
| 3 | 36966266 | 40257154 | indirect |
| 4 | 23018048 | 29306473 | indirect |
| 0 | 25441021 | 45652298 | isolated |
| 1 | 42996077 | 35741584 | isolated |
| 2 | 25335343 | 22510809 | isolated |
| 3 | 25763556 | 2081159 | isolated |
| 4 | 40324266 | 33151168 | isolated |

## 5.2 Quantitative Analysis

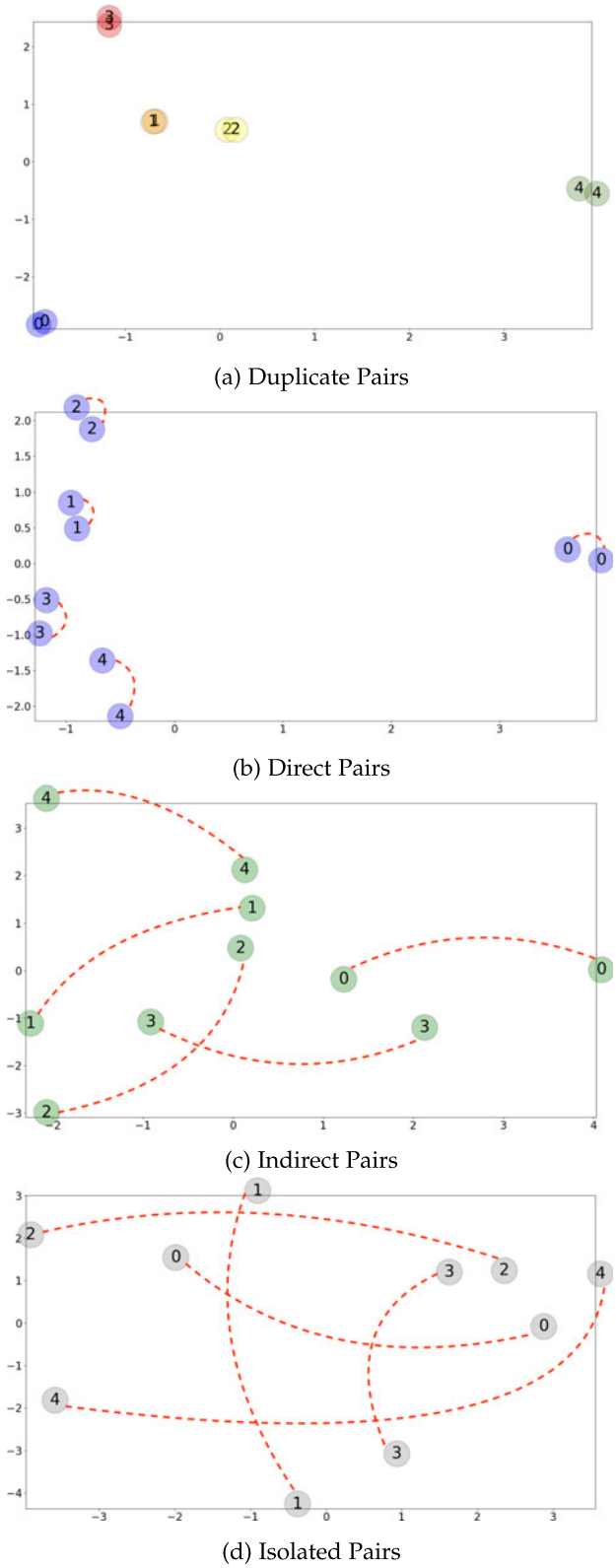
### 5.2.1 Stability of Post2Vec Representation

We find that the performance rankings of different post representations for different downstream tasks are not always consistent. Take TagCNN as an example, it performs second-best in relatedness prediction but performs worst in post classification and API recommendation. Moreover, after integrating the post representation learned by D2V and TagCNN into the state-of-the-art approaches, the performance becomes worse for the post-classification (see Table 5) and API recommendation (see Table 6) tasks. It demonstrates that the performance could be even harmed when integrating unsuitable post representation. Overall, the experimental result demonstrates that the post representation learned by Post2Vec is not only higher quality but also more stable than others.

### 5.2.2 Robustness of Post2Vec Representation

Recall that there are four handcrafted features proposed in the original work of relatedness prediction and we append the distance between posts' vectors generated by Post2Vec (denoted by $P2V_{Dist}$) to boost the features (Section 4.2.4). In this way, we construct a five-dimensional feature vector ($FV$) for each pair of posts, i.e., $\langle cos_{SO}, soft_{SO}, soft_{Google}, soft_{Edit}, P2V_{Dist}\rangle$.

To measure the relative importance of the feature generated by Post2Vec, we remove each component of $FV$ one-at-a-time and measure the corresponding performance (in terms of F1-score; which is the summary metric). The most important feature should result in the largest drop in performance when it is removed. As shown in Table 9, we find that F1-score decreases the most when Dist (the feature generated by Post2Vec) is removed. This shows that the representation learned by Post2Vec is the most important feature among the five. The rank of the relative importance of the five features is $P2V_{Dist} > soft_{SO} > soft_{Google} > cos_{SO} = soft_{Edit}$.



(a) Duplicate Pairs

(b) Direct Pairs

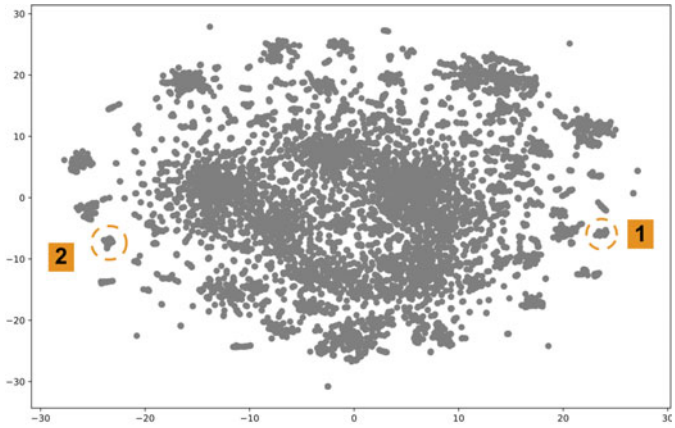
(c) Indirect Pairs

(d) Isolated Pairs

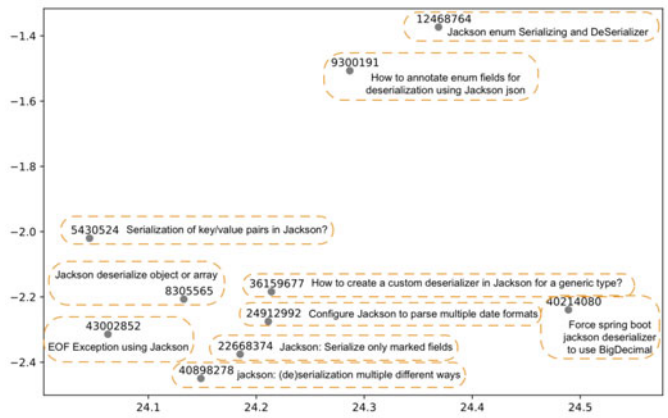
Fig. 4. Vector space of Post2Vec on relatedness prediction.

## 5.3 Vector Space Visualization

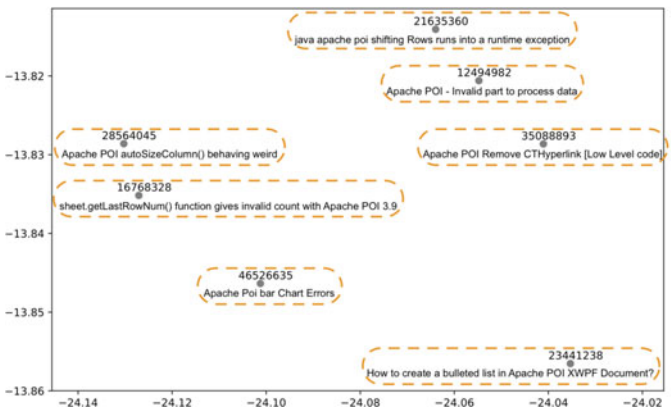
To further investigate the clustering quality of post vectors learned by Post2Vec, we randomly sample a subset of posts with a significant number from the dataset of relatedness prediction and visualize the vector space learned by Post2Vec. Out of the 51,918 posts from the dataset, we randomly




(a) Vector Space




(b) An *Jackson*-related Cluster (#1) of Posts in Figure 5a




(c) An *Apache POI*-related Cluster (#2) of Posts in Figure 5a

*: The numbers in Figure 5b and 5c represent posts' Ids in Stack Overflow.

Fig. 5. Vector space learned by Post2Vec.

consider 8,105 posts (i.e., the exact number provided by the Sample Size Calculator[11] using 95 percent for the confidence level). Different from the vector space visualization for the relatedness prediction in Section 5.1, we omit the relatedness labels between posts. As shown in Fig. 5, we utilize the t-SNE dimensionality reduction technique [55] to visualize the vector space learned by our approach. We find that the learned vector space contains different sizes of clusters. Furthermore, we arbitrarily pick two clusters (as shown in Figs. 5b and 5c) and manually read all the included posts. We find that posts in a cluster are related as they are mainly about one or a few specific technologies. For example, all the posts in cluster #1 (i.e., Fig. 5b) are related to a Java API named *Jackson* while most of the posts (except one) in cluster #2 are related to another API called *Apache POI*. To some extent, it also explains that the boundaries between clusters are not always clear as the posts may be related to multiple technologies instead of only one.

Within a cluster, we find that the closer two posts' vectors are, the more sentiment similar they are. Take a pair of posts in cluster #1 as an example (i.e., Fig. 5b), the two posts (Id = 9300191 and 12468764) are close to each other in the vector space but far from others. The reason is that only these two posts are related to deserializing enumeration by using Jackson. The same observation can also be found in cluster #2 which is related to another technology named *Apache POI*. Within cluster #2, the pair of posts that most close to each other (i.e., Id = 28564045 and 16768328) are related to version 3.9 of the API *Apache POI* and the usage of API function calls for editing sheets. Above all, two posts' vectors are likely to be close to each other in the vector space learned by our approach if they are similar semantically.

### 5.4 Threats to Validity

A threat to internal validity relates to our assumption that tags are labeled correctly by users in Stack Overflow; however, mislabeling could happen sometimes. Still, we believe that Stack Overflow effective crowdsourcing process helps to reduce the number of such cases. Also, some works utilize a validation dataset to tune hyperparameters during training to reduce overfitting likelihood. However, similar to many prior works [56], [57], for efficiency reason, we did not tune the hyperparameters. Tuning them can potentially boost Post2Vec effectiveness. Still, the experimental results have demonstrated that Post2Vec can improve the effectiveness of state-of-the-art approaches in several tasks, which suggests that the learned post representation does not overfit training data.

One threat to external validity relates to the generality of the distributed representation generated by Post2Vec. Related works like fun2vec [58] and code2vec [59], that aim to generate a representation for functions and methods, only consider at most one downstream task. In this work, we have reduced this threat by considering three downstream tasks. Another threat to external validity is that the effectiveness of our distributed representation depends on the richness of the tags. We observed that there are more than 58k tags in Stack Overflow on August 2019 and on average every post is labeled with 3 tags. Moreover, Stack Overflow encourages users to label their questions with existing tags and requires them to avoid meta-tags (i.e., generic tags that do not describe the content of the question, e.g., *beginners*) when creating new tags.[12] The aforementioned facts suggest that the tags in Stack Overflow are commonly used to characterize posts and they are meaningful for capturing the contents of posts. In the future, we will explore other potential information (in addition to the tags)

11. https://www.surveysystem.com/sscalc.htm

12. Users guide on tags in Stack Overflow, https://stackoverflow.com/help/tagging.

TABLE 9
Related Importance of Different Combinations of Features for Relatedness Prediction

| | Feature Vector | Duplicate | Direct | Indirect | Isolated | Overall |
|---|---|---|---|---|---|---|
| Precision | $FV_{w/o\ cos_{SO}}$ | 0.53 | 0.45 | 0.48 | 0.77 | 0.55 |
| | $FV_{w/o\ soft_{SO}}$ | 0.53 | 0.45 | 0.46 | 0.69 | 0.53 |
| | $FV_{w/o\ soft_{Google}}$ | 0.53 | 0.44 | 0.48 | 0.76 | 0.55 |
| | $FV_{w/o\ soft_{Edit}}$ | 0.53 | 0.45 | 0.48 | 0.77 | 0.55 |
| | $FV_{w/o\ P2V_{Dist}}$ | 0.47 | 0.41 | 0.43 | 0.73 | 0.51 |
| | $FV_{all}$ | 0.53 | 0.45 | 0.48 | 0.77 | **0.56** |
| Recall | $FV_{w/o\ cos_{SO}}$ | 0.60 | 0.31 | 0.48 | 0.91 | 0.57 |
| | $FV_{w/o\ soft_{SO}}$ | 0.60 | 0.32 | 0.38 | 0.93 | 0.55 |
| | $FV_{w/o\ soft_{Google}}$ | 0.61 | 0.31 | 0.47 | 0.91 | 0.57 |
| | $FV_{w/o\ soft_{Edit}}$ | 0.60 | 0.30 | 0.48 | 0.91 | 0.57 |
| | $FV_{w/o\ P2V_{Dist}}$ | 0.54 | 0.19 | 0.49 | 0.91 | 0.53 |
| | $FV_{all}$ | 0.61 | 0.31 | 0.48 | 0.91 | **0.58** |
| F1-score | $FV_{w/o\ cos_{SO}}$ | 0.56 | 0.37 | 0.48 | 0.83 | 0.56 |
| | $FV_{w/o\ soft_{SO}}$ | 0.57 | 0.37 | 0.42 | 0.79 | 0.53 |
| | $FV_{w/o\ soft_{Google}}$ | 0.57 | 0.37 | 0.47 | 0.83 | 0.55 |
| | $FV_{w/o\ soft_{Edit}}$ | 0.56 | 0.36 | 0.48 | 0.83 | 0.56 |
| | $FV_{w/o\ P2V_{Dist}}$ | 0.51 | 0.26 | 0.46 | 0.81 | 0.51 |
| | $FV_{all}$ | 0.57 | 0.36 | 0.48 | 0.83 | **0.56** |

that can also be used to guide the representation learning process.

Threats to construct validity relate to the evaluation metrics and the statistical hypothesis test that we consider. We reuse the evaluation metrics that were used in the original tag prediction, relatedness prediction, post classification, and API recommendation work that we extend. We use a standard statistical hypothesis test, Wilcoxon signed-rank test [49], to check whether the performance difference between two competing approaches is significant. This test has been used in many past studies, e.g., [60], [61], [62]. Most of the metrics and the statistical hypothesis test are well known. Thus, we believe that this threat is minimal.

## 6 Related Work

In this section, we describe prior studies on SQA post analysis, and those that learn distributed representations of software artifacts.

### 6.1 SQA Post Analysis

Many past studies have analyzed SQA posts to help developers find relevant information on SQA sites; these include studies on tag recommendation for posts [1], [2], [3], relatedness prediction for posts [4], [6], relevant question retrieval [41], [63], [64], post classification [8], [9], [10], etc. Moreover, SQA posts have also been analyzed to help developers in performing other tasks such as code summarization [65], [66], [67], and code comprehension [68], [69], [70], etc. We briefly describe some of the aforementioned works below.

#### 6.1.1 Tag Recommendation

Xia *et al.* proposed TagCombine, a method that analyzes mappings between posts and tags from different perspectives [1]. TagCombine used bag-of-words (BOW) to represent posts. The representation was fed into three components: multi-label ranking, similarity based ranking, and tag-term based ranking. The combination of scores produced by the 3 components were used to rank and recommend tags to a post. Recently, Zhou *et al.* have proposed four different deep learning approaches, TagCNN, TagRNN, TagHAN and TagRCNN, which are based on convolutional neural networks (CNN), recurrent neural networks (RNN), hierarchical attention networks (HAN), and recurrent convolutional neural network (RCNN), respectively [3]. All four approaches concatenate the text in the title and description of posts as the input data to predict the posts' tags. Their experimental results show that TagCNN and TagRCNN achieve the best performance, and outperform the aforementioned TagCombine proposed by Xia *et al.* [1] and several other baselines.

#### 6.1.2 Related Post Prediction

Xu *et al.* had come up with four relatedness types for measuring relationship between posts and formulated the relatedness prediction problem as a multi-class classification problem [4]. Fu *et al.* extended Xu *et al.*'s work using a support vector machine (SVM) called TunedSVM [5]. Compared with Xu *et al.*'s original approach, TunedSVM achieved similar effectiveness but was much faster. In their latest work, Xu *et al.* proposed four new handcrafted features and a more efficient SVM model, namely SoftSVM [6]. The results showed that SoftSVM achieved better performance in terms of both effectiveness and efficiency.

#### 6.1.3 Post Classification

Allamanis *et al.* employed a topic model for clustering questions in Stack Overflow into categories [9]. They defined a category as "the set of reasons questions are asked and what the users are trying to accomplish". They found that question category does not vary for different programming languages. Very recently, Beyer *et al.* did a study focusing

on Android-related Stack Overflow posts to investigate question categories, and proposed an approach that can automatically classify the posts into these categories [10]. More specifically, they manually labeled 500 posts with seven categories, API CHANGE, API USAGE, CONCEPTUAL, DISCREPANCY, LEARNING, ERRORS, and REVIEW. Then, they used these labeled posts for building models to automate post classification.

Post2Vec can aid the aforementioned works by learning a better representation of Stack Overflow posts. In Section 3, we have demonstrated that Post2Vec can perform better than the state-of-the-art approaches proposed by Zhou *et al.* [3] for tag recommendation. Additionally, in Section 4, we have also shown that Post2Vec's distributed representations can boost the performance of related post prediction, and post classification.

### 6.2 Representations of Software Artifacts

Recently, a number of approaches have been proposed to learn distributed representation of software artifacts [58], [59], [71]. For example, DeFreez *et al.* proposed Func2vec, which learns to map a code function to a vector in continuous vector space [58]. They first linearized code of a function to generate path sequences as sentences, which were generated using random walks over the paths in the control-flow graph of the code. On these sentences, a neural network was trained to learn function embedding. Also, Alon *et al.* proposed code2vec, which learns *code embeddings* - a continuously distributed vector representation of code snippets [59].

Narayanan *et al.* proposed apk2vec, which uses a graph embedding approach to build distributed representations of android apps [71]. Their approach consisted of a static analysis phase which took in an APK file and generated dependency graphs from the code in the file. The information from the dependency graphs was then fed into a neural network which subsequently combined them and generated a distributed representation of the APK file.

Different from the aforementioned works, our target is to learn the distributed representations of software posts, rather than source code or APKs. Our work can potentially be enhanced by making use of the code representations proposed by DeFreez *et al.* and Alon *et al.* which we leave for future work. One benefit of our approach is that it is programming language agnostic, unlike the representations proposed by DeFreez *et al.* and Alon *et al.* This makes our approach suitable for corpus containing a mix of code snippets in different programming languages, such as the code snippets in the Stack Overflow posts used in this work.

## 7 Conclusion and Future Work

In this work, we present a novel neural network based architecture, namely Post2Vec, which learns distributed representation of Stack Overflow posts by using the tags assigned by users to such posts as a guide. Post2Vec architecture takes advantage of code snippets and handles different components (i.e., title, description, and code snippets) separately.

To evaluate the quality of Post2Vec's deep learning architecture, we first investigate its end-to-end effectiveness in tag recommendation task. The experimental results showed that Post2Vec outperforms the best performing state-of-the-art deep learning based tag recommendation approaches, suggesting its ability to learn better post representations. Furthermore, to evaluate the value of representation learned using Post2Vec, we integrated the generated post representations into the learning pipeline of three downstream tasks, i.e., relatedness prediction, post classification, and API recommendation. We found that the post representation learned using Post2Vec can boost the effectiveness of state-of-the-art approaches' performance by a substantial margin.

In the future, we would like to explore the potential improvements of post representation by integrating more information, e.g., comments and answers. We also plan to investigate the effectiveness of other deep neural network models (beyond CNN and RNN) to be used in the feature extraction layer of Post2Vec. Moreover, we plan to further investigate the effectiveness of post representation learned by Post2Vec in additional downstream tasks. It would also be interesting to investigate the applicability of Post2Vec to additional software question and answer sites beyond Stack Overflow.

## Acknowledgments

This work was supported by the Ministry of Education, Singapore, under its Academic Research Fund Tier 2 Award No. MOE2019-T2-1-193. Any opinions, findings, and conclusions, or recommendations expressed in this material are those of the author(s) and do not reflect the views of the Ministry of Education, Singapore.

## References

[1] X. Xia, D. Lo, X. Wang, and B. Zhou, "Tag recommendation in software information sites," in *Proc. 10th Work. Conf. Mining Softw. Repositories*, 2013, pp. 287–296.

[2] S. Wang, D. Lo, B. Vasilescu, and A. Serebrenik, "EnTagRec: An enhanced tag recommendation system for software information sites," in *Proc. IEEE Int. Conf. Softw. Maintenance Evol.*, 2014, pp. 291–300.

[3] P. Zhou, J. Liu, X. Liu, Z. Yang, and J. Grundy, "Is deep learning better than traditional approaches in tag recommendation for software information sites?," *Inf. Softw. Technol.*, 2019, pp. 1–13.

[4] B. Xu, D. Ye, Z. Xing, X. Xia, G. Chen, and S. Li, "Predicting semantically linkable knowledge in developer online forums via convolutional neural network," in *Proc. 31st IEEE/ACM Int. Conf. Automated Softw. Eng.*, 2016, pp. 51–62.

[5] W. Fu and T. Menzies, "Easy over hard: A case study on deep learning," in *Proc. 11th Joint Meeting Found. Softw. Eng.*, 2017, pp. 49–60.

[6] B. Xu, A. Shirani, D. Lo, and M. A. Alipour, "Prediction of relatedness in stack overflow: Deep learning versus SVM: A reproducibility study," in *Proc. 12th ACM/IEEE Int. Symp. Empir. Softw. Eng. Meas.*, 2018, Art. no. 21.

[7] T. Menzies, S. Majumder, N. Balaji, K. Brey, and W. Fu, "500+ times faster than deep learning:(A case study exploring faster methods for text mining stackoverflow)," in *Proc. IEEE/ACM 15th Int. Conf. Mining Softw. Repositories*, 2018, pp. 554–563.

[8] C. Rosen and E. Shihab, "What are mobile developers asking about? A large scale study using stack overflow," *Empir. Softw. Eng.*, vol. 21, no. 3, pp. 1192–1223, 2016.

[9] M. Allamanis and C. Sutton, "Why, when, and what: Analyzing stack overflow questions by topic, type, and code," in *Proc. 10th Work. Conf. Mining Softw. Repositories.*., 2013, pp. 53–56.

[10] S. Beyer, C. Macho, M. Pinzger, and M. D. Penta, "Automatically classifying posts into question categories on stack overflow," in *Proc. 26th Conf. Program Comprehension*, 2018, pp. 211–221.

[11] C. Treude and M. Wagner, "Predicting good configurations for GitHub and stack overflow topic models," in *Proc. IEEE/ACM 16th Int. Conf. Mining Softw. Repositories.*, 2019, pp. 84–95.

[12] X.-L. Yang, D. Lo, X. Xia, Z.-Y. Wan, and J.-L. Sun, "What security questions do developers ask? A large-scale study of stack overflow posts," *J. Comput. Sci. Technol.*, vol. 31, no. 5, pp. 910–924, 2016.

[13] S. Wang, D. Lo, B. Vasilescu, and A. Serebrenik, "EnTagRec++: An enhanced tag recommendation system for software information sites," *Empir. Softw. Eng.*, vol. 23, no. 2, pp. 800–832, 2018.

[14] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2013, pp. 3111–3119.

[15] B. Xu, Z. Xing, X. Xia, and D. Lo, "AnswerBot: Automated generation of answer summary to developersź technical questions," in *Proc. 32nd IEEE/ACM Int. Conf. Automated Softw. Eng.*, 2017, pp. 706–716.

[16] Q. Le and T. Mikolov, "Distributed representations of sentences and documents," in *Proc. Int. Conf. Mach. Learn.*, 2014, pp. 1188–1196.

[17] Q. Huang, X. Xia, Z. Xing, D. Lo, and X. Wang, "API method recommendation without worrying about the task-API knowledge gap," in *Proc. 33rd IEEE/ACM Int. Conf. Automated Softw. Eng.*, 2018, pp. 293–304.

[18] X. Li, Y. Meng, X. Sun, Q. Han, A. Yuan, and J. Li, "Is word segmentation necessary for deep learning of chinese representations?," in *Proc. 57th Conf. Assoc. Comput. Linguistics*, 2019, pp. 3242–3252.

[19] M. White, M. Tufano, C. Vendome, and D. Poshyvanyk, "Deep learning code fragments for code clone detection," in *Proc. 31st IEEE/ACM Int. Conf. Automated Softw. Eng.*, 2016, pp. 87–98.

[20] M. Tufano, C. Watson, G. Bavota, M. D. Penta, M. White, and D. Poshyvanyk, "Deep learning similarities from different representations of source code," in *Proc. IEEE/ACM 15th Int. Conf. Mining Softw. Repositories*, 2018, pp. 542–553.

[21] S. Chakraborty, Y. Ding, M. Allamanis, and B. Ray, "CODIT: Code editing with tree-based neural models," *IEEE Trans. Softw. Eng.*, early access, Aug. 31, 2020, doi: 10.1109/TSE.2020.3020502.

[22] W. Yin, K. Kann, M. Yu, and H. Schütze, "Comparative study of cnn and RNN for natural language processing," 2017, *arXiv: 1702.01923*.

[23] J. Han and C. Moraga, "The influence of the sigmoid function parameters on the speed of backpropagation learning," in *Proc. Int. Workshop Artif. Neural Netw.*, 1995, pp. 195–201.

[24] R. Caruana, S. Lawrence, and C. L. Giles, "Overfitting in neural nets: Backpropagation, conjugate gradient, and early stopping," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2001, pp. 402–408.

[25] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *J. Mach. Learn. Res.*, vol. 15, no. 1, pp. 1929–1958, 2014.

[26] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *Proc. 3rd Int. Conf. Learn. Representations*, 2015.

[27] M. Anthimopoulos, S. Christodoulidis, L. Ebner, A. Christe, and S. Mougiakakou, "Lung pattern classification for interstitial lung diseases using a deep convolutional neural network," *IEEE Trans. Med. Imag.*, vol. 35, no. 5, pp. 1207–1216, May 2016.

[28] S. Arora, N. Cohen, and E. Hazan, "On the optimization of deep networks: Implicit acceleration by overparameterization," in *Proc. 35th Int. Conf. Mach. Learn.*, 2018, pp. 244–253. [Online]. Available: http://proceedings.mlr.press/v80/arora18a.html

[29] M. T. Hagan and M. B. Menhaj, "Training feedforward networks with the marquardt algorithm," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 5, no. 6, pp. 989–993, Nov. 1994.

[30] C. Treude and M. P. Robillard, "Understanding stack overflow code fragments," in *Proc. IEEE Int. Conf. Softw. Maintenance Evol.*, 2017, pp. 509–513.

[31] Y. Kim, "Convolutional neural networks for sentence classification," in *Proc. Conf. Empir. Methods Natural Lang. Process.*, 2014, pp. 1746–1751. [Online]. Available: http://aclweb.org/anthology/D/D14/D14–1181.pdf

[32] P. Liu, X. Qiu, and X. Huang, "Recurrent neural network for text classification with multi-task learning," in *Proc. 25th Int. Joint Conf. Artif. Intell.*, 2016, pp. 2873–2879. [Online]. Available: http://www.ijcai.org/Abstract/16/408

[33] Z. Yang, D. Yang, C. Dyer, X. He, A. Smola, and E. Hovy, "Hierarchical attention networks for document classification," in *Proc. Conf. North Amer. Chapter Assoc. Comput. Linguistics Hum. Lang. Technol.*, 2016, pp. 1480–1489.

[34] S. Lai, L. Xu, K. Liu, and J. Zhao, "Recurrent convolutional neural networks for text classification," in *Proc. 29th AAAI Conf. Artif. Intell.*, 2015, pp. 2267–2273.

[35] S. Wang, D. Lo, B. Vasilescu, and A. Serebrenik, "EnTagRec: An enhanced tag recommendation system for software information sites," in *Proc. 30th IEEE Int. Conf. Softw. Maintenance Evol.*, 2014, pp. 291–300.

[36] P. Zhou, J. Liu, Z. Yang, and G. Zhou, "Scalable tag recommendation for software information sites," in *Proc. IEEE 24th Int. Conf. Softw. Anal., Evol. Reeng.*, 2017, pp. 272–282.

[37] J. Liu, P. Zhou, Z. Yang, X. Liu, and J. Grundy, "FastTagRec: Fast tag recommendation for software information sites," *Automated Softw. Eng.*, vol. 25, no. 4, pp. 675–701, 2018.

[38] Z. Gao, X. Xia, J. Grundy, D. Lo, and Y.-F. Li, "Generating question titles for stack overflow from mined code snippets," *ACM Trans. Softw. Eng. Methodol.*, vol. 29, no. 4, pp. 1–37, Sep. 2020.

[39] L. Wang, L. Zhang, and J. Jiang, "Detecting duplicate questions in stack overflow via deep learning approaches," in *Proc. 26th Asia-Pacific Softw. Eng. Conf.*, 2019, pp. 506–513.

[40] L. Ponzanelli, A. Mocci, A. Bacchelli, M. Lanza, and D. Fullerton, "Improving low quality stack overflow post detection," in *Proc. IEEE Int. Conf. Softw. Maintenance Evol.*, 2014, pp. 541–544.

[41] 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 *Proc. 38th Int. Conf. Softw. Eng.*, 2016, pp. 404–415.

[42] L. Ponzanelli, G. Bavota, M. D. Penta, R. Oliveto, and M. Lanza, "Mining stackoverflow to turn the IDE into a self-confident programming prompter," in *Proc. 11th Work. Conf. Mining Softw. Repositories*, 2014, pp. 102–111.

[43] M. Li, T. Zhang, Y. Chen, and A. J. Smola, "Efficient mini-batch training for stochastic optimization," in *Proc. 20th ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2014, pp. 661–670.

[44] V. J. Hellendoorn, P. T. Devanbu, and M. A. Alipour, "On the naturalness of proofs," in *Proc. 26th ACM Joint Meeting Eur. Softw. Eng. Conf. Symp. Found. Softw. Eng.*, 2018, pp. 724–728.

[45] H. Ha and H. Zhang, "DeepPerf: Performance prediction for configurable software with deep sparse neural network," in *Proc. IEEE/ACM 41st Int. Conf. Softw. Eng.*, 2019, pp. 1095–1106.

[46] J. Chen *et al.* "Unblind your apps: Predicting natural-language labels for mobile GUI components by deep learning," in *Proc. IEEE/ACM 42nd Int. Conf. Softw. Eng.*, 2020, pp. 322–334.

[47] X. Gu, H. Zhang, and S. Kim, "Deep code search," in *Proc. IEEE/ACM 40th Int. Conf. Softw. Eng.*, 2018, pp. 933–944.

[48] Y.-L. Boureau, J. Ponce, and Y. LeCun, "A theoretical analysis of feature pooling in visual recognition," in *Proc. 27th Int. Conf. Mach. Learn.*, 2010, pp. 111–118.

[49] F. Wilcoxon, "Individual comparisons by ranking methods," in *Breakthroughs in Statistics*. Berlin, Germany: Springer, 1992, pp. 196–202.

[50] R. Řehůřek and P. Sojka, "Software Framework for Topic Modelling with Large Corpora," in *Proc. LREC Workshop New Challenges NLP Frameworks*, 2010, pp. 45–50. [Online]. Available: http://is.muni.cz/publication/884893/en

[51] C. Rorres and H. Anton, *Elementary Linear Algebra: Applications Version*. Hoboken, NJ, USA: Wiley, 1994.

[52] C. Kadilar and H. Cingi, "Ratio estimators in stratified random sampling," *Biometrical J. Math. Methods Biosci.*, vol. 45, no. 2, pp. 218–225, 2003.

[53] M. M. Rahman, C. K. Roy, and D. Lo, "RACK: Automatic API recommendation using crowdsourced knowledge," in *Proc. IEEE 23rd Int. Conf. Softw. Anal., Evol., Reeng.*, 2016, pp. 349–359.

[54] X. Gu, H. Zhang, D. Zhang, and S. Kim, "Deep API learning," in *Proc. 24th ACM SIGSOFT Int. Symp. Found. Softw. Eng.*, 2016, pp. 631–642.

[55] L. V. D. Maaten and G. Hinton, "Visualizing data using t-SNE," *J. Mach. Learn. Res.*, vol. 9, no. Nov, pp. 2579–2605, 2008.

[56] T. Hoang, H. K. Dam, Y. Kamei, D. Lo, and N. Ubayashi, "DeepJIT: An end-to-end deep learning framework for just-in-time defect prediction," in *Proc. 16th Int. Conf. Mining Softw. Repositories*, 2019, pp. 34–45.

[57] X. Huo and M. Li, "Enhancing the unified features to locate buggy files by exploiting the sequential nature of source code," in *Proc. 26th Int. Joint Conf. Artif. Intell.*, 2017, pp. 1909–1915.
[58] D. DeFreez, A. V. Thakur, and C. Rubio-Gonz ález, "Path-based function embedding and its application to error-handling specification mining," in *Proc. 26th ACM Joint Meeting Eur. Softw. Eng. Conf. Symp. Found. Softw. Eng.*, 2018, pp. 423–433.
[59] U. Alon, M. Zilberstein, O. Levy, and E. Yahav, "code2vec: Learning distributed representations of code," *Proc. ACM Program. Lang.*, vol. 3, 2019, Art. no. 40.
[60] J. Jiarpakdee, C. Tantithamthavorn, H. K. Dam, and J. Grundy, "An empirical study of model-agnostic techniques for defect prediction models," *IEEE Trans. Softw. Eng.*, early access, Mar. 23, 2020, doi: 10.1109/TSE.2020.2982385.
[61] G. K. Rajbahadur, S. Wang, Y. Kamei, and A. E. Hassan, "Impact of discretization noise of the dependent variable on machine learning classifiers in software engineering," *IEEE Trans. Softw. Eng.*, early access, Jun. 24, 2019, doi: 10.1109/TSE.2019.2924371.
[62] S. Wang, T. Liu, J. Nam, and L. Tan, "Deep semantic feature learning for software defect prediction," *IEEE Trans. Softw. Eng.*, vol. 46, no. 12, pp. 1267–1293, Dec. 2020.
[63] B. Xu, Z. Xing, X. Xia, D. Lo, Q. Wang, and S. Li, "Domain-specific cross-language relevant question retrieval," in *Proc. 13th Int. Conf. Mining Softw. Repositories*, 2016, pp. 413–424.
[64] Y. Zou, T. Ye, Y. Lu, J. Mylopoulos, and L. Zhang, "Learning to rank for question-oriented software text retrieval (t)," in *Proc. 30th IEEE/ACM Int. Conf. Automated Softw. Eng.*, 2015, pp. 1–11.
[65] E. Wong, J. Yang, and L. Tan, "Autocomment: Mining question and answer sites for automatic comment generation," in *Proc. 28th IEEE/ACM Int. Conf. Automated Softw. Eng.*, 2013, pp. 562–567.
[66] L. Guerrouj, D. Bourque, and P. C. Rigby, "Leveraging informal documentation to summarize classes and methods in context," in *Proc. IEEE/ACM 37th IEEE Int. Conf. Softw. Eng.*, 2015, pp. 639–642.
[67] L. Ponzanelli, A. Mocci, and M. Lanza, "Summarizing complex development artifacts by mining heterogeneous data," in *Proc. 12th Work. Conf. Mining Softw. Repositories*, 2015, pp. 401–405.
[68] D. Yang, A. Hussain, and C. V. Lopes, "From query to usable code: An analysis of stack overflow code snippets," in *Proc. 13th Int. Conf. Mining Softw. Repositories*, 2016, pp. 391–402.
[69] T. Diamantopoulos and A. Symeonidis, "Employing source code information to improve question-answering in stack overflow," in *Proc. IEEE/ACM 12th Work. Conf. Mining Softw. Repositories*, 2015, pp. 454–457.
[70] S. Subramanian and R. Holmes, "Making sense of online code snippets," in *Proc. 10th Work. Conf. Mining Softw. Repositories*, 2013, pp. 85–88.
[71] A. Narayanan, C. Soh, L. Chen, Y. Liu, and L. Wang, "apk2vec: Semi-supervised multi-view representation learning for profiling android applications," in *Proc. IEEE Int. Conf. Data Mining*, 2018, pp. 357–366.
[72] J. Zhou, H. Zhang, and D. Lo, "Where should the bugs be fixed? More accurate information retrieval-based bug localization based on bug reports," in *Proc. 34th Int. Conf. Softw. Eng.*, 2012, pp. 14–24.
[73] R. K. Saha, M. Lease, S. Khurshid, and D. E. Perry, "Improving bug localization using structured information retrieval," in *Proc. 28th IEEE/ACM Int. Conf. Automated Softw. Eng.*, 2013, pp. 345–355.
[74] G. Chen, C. Chen, Z. Xing, and B. Xu, "Learning a dual-language vector space for domain-specific cross-lingual question retrieval," in *Proc. 31st IEEE/ACM Int. Conf. Automated Softw. Eng.*, 2016, pp. 744–755.
[75] P. S. Kochhar, T.-D. B. Le, and D. Lo, "It's not a bug, it's a feature: Does misclassification affect bug localization?," in *Proc. 11th Work. Conf. Mining Softw. Repositories*, 2014, pp. 296–299.

**Bowen Xu** received the MEng degree from the College of Software Technology, Zhejiang University, China, in 2017. He is currently working toward the PhD degree at the School of Computing and Information Systems, Singapore Management University, Singapore. His research to date has focused on mining and analyzing rich data in software repositories to uncover interesting and actionable knowledge. For more information, please visit http://www.bowenxu.me.

**Thong Hoang** received the BEng and MS degrees in computer science from the University of Technology - Vietnam and Konkuk University - South Korea, respectively, and the PhD degree from the School of Computing and Information System, Singapore Management University advised by Professor David Lo. He was a research fellow with the Living Analytics Research Centre (LARC) in 2020–2021. His research interests include software bug localization, defect prediction, deep learning, and data mining area.

**Abhishek Sharma** received the PhD degree in information systems from the School of Computing and Information Systems, Singapore Management University. He is currently a machine learning engineer with Veracode, Singapore. His research interests include software analytics, mining software repositories, social network mining, and empirical software engineering. For more information, please visit https://abhishek9sharma.github.io.

**Chengran Yang** is currently a research engineer with the School of Computing and Information Systems, Singapore Management University, Singapore. His research interests include the intersection between software engineering and artificial intelligence.

**Xin Xia** received the PhD degree in computer science from Zhejiang University, China, in 2014. He is the director of the software engineering application technology lab, Huawei, China. Prior to joining Huawei, he was an ARC DECRA Fellow and a lecturer with Monash University, Australia. To help developers and testers improve their productivity, his current research focuses on mining and analyzing rich data in software repositories to uncover interesting and actionable information. For more information, please visit https://xinxia.github.io.

**David Lo** received the PhD degree in computer science from the National University of Singapore. He is a professor of Computer Science with the School of Computing and Information Systems, Singapore Management University. His research interests include software analytics, software maintenance, software testing, social network mining, and cybersecurity. He is an ACM Distinguished Member and the 2021 recipient of the IEEE TCSE Distinguished Service Award. For more information, please visit http://www.mysmu.edu/faculty/davidlo.