5-2022

Benchmarking library recognition in tweets

Ting ZHANG

Divya Prabha CHANDRASEKARAN

Ferdian THUNG

David LO
Singapore Management University, davidlo@smu.edu.sg

Benchmarking Library Recognition in Tweets

Ting Zhang, Divya Prabha Chandrasekaran, Ferdian Thung, David Lo
School of Computing and Information Systems, Singapore Management University
{tingzhang.2019,divyaprabha.2021,ferdianthung,davidlo}@smu.edu.sg

ABSTRACT

Software developers often use social media (such as Twitter) to share programming knowledge such as new tools, sample code snippets, and tips on programming. One of the topics they talk about is the software library. The tweets may contain useful information about a library. A good understanding of this information, e.g., on the developer's views regarding a library can be beneficial to weigh the pros and cons of using the library as well as the general sentiments towards the library. However, it is not trivial to recognize whether a word actually refers to a library or other meanings. For example, a tweet mentioning the word "`pandas`" may refer to the Python pandas library or to the animal. In this work, we created the first benchmark dataset and investigated the task to distinguish whether a tweet refers to a programming library or something else. Recently, the pre-trained Transformer models (PTMs) have achieved great success in the fields of natural language processing and computer vision. Therefore, we extensively evaluated a broad set of modern PTMs, including both general-purpose and domain-specific ones, to solve this programming library recognition task in tweets. Experimental results show that the use of PTM can outperform the best-performing baseline methods by 5% - 12% in terms of F1-score under within-, cross-, and mixed-library settings.

CCS CONCEPTS

• **Software and its engineering → Software libraries and repositories**.

KEYWORDS

software libraries, tweets, disambiguation, benchmark study

ACM Reference Format:
Ting Zhang, Divya Prabha Chandrasekaran, Ferdian Thung, David Lo. 2022. Benchmarking Library Recognition in Tweets. In *30th International Conference on Program Comprehension (ICPC '22), May 16–17, 2022, Virtual Event, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3524610.3527916

1 INTRODUCTION

With the proliferation of social media, such as Twitter, people express more and more opinions and share knowledge online. Software developers also widely use Twitter to stay current, for learning, and for building relationships [32]. Tweets can have valuable information to help software developers [13]. For example, research has shown that tweets can be helpful for requirement engineers to better understand user needs, thereby providing important information for software evolution [14, 15]. Motivated by the impact of tweets for software development, there has been much research on tweets for software engineering (SE) in the past years [10, 33, 44]. Most of the existing research focused on classifying tweets into broad categories (e.g., classify if a tweet is software-related or not) [7, 30].

Recent years have witnessed a boom of third-party libraries/-packages. Take Python libraries for example, as of 10 Jan 2022, there are 349,177 publicly available libraries.[1] Leveraging these libraries can greatly reduce software development efforts. However, it is not trivial to find users' opinions regarding certain libraries, which would help developers in choosing the libraries. Twitter is one of the platforms where people share their opinions about libraries. Take two tweets as examples: "`boto` *is the best way to interface with Amazon Web Services when using Python*" and "*Turns out that boto3 instead of* `boto` *makes a big difference*"; they contain valuable information about the usefulness of `boto` and the version update. Those who want to interface with Amazon Web Services or struggle to use `boto` or `boto3` can find these tweets useful. Being able to harvest the library information from tweets may help developers to select and learn about a particular library for their tasks. Certainly, a library that receives generally negative reviews on Twitter may not be an ideal choice. A lack of research on library-related tweets motivates us to make the first attempt. In this paper, we attempt to filter useful library-related tweets to help library developers and users. We hope our research can encourage follow-up research on utilizing the information in tweets for library development or recommendation: Library users can identify suitable libraries to help their software development effort, while library developers can leverage these library-related tweets as feedback for improving their libraries.

Extracting library-related tweets out of a large volume of tweets is challenging. A significant challenge in extracting library-related tweets is the limited size of the text in a tweet. Compared to other SE artifacts such as posts in Stack Overflow and Stack Exchange, tweets are shorter in length. This makes extracting meaningful information from them comparatively more difficult. Another challenge is to disambiguate the *library sense* (an occurrence of a word that refers to a specific library) from its normal sense (an occurrence of a word that does not refer to the specific library). Take `flask` as an example: its library sense refers to a third-party Python library for developing web applications, and its normal sense is a container for liquids. Library recognition in tweets involves classifying a tweet as related to a *specific* library or not. When a user searches for a

ICPC '22, May 16–17, 2022, Virtual Event, USA

ACM ISBN 978-1-4503-9298-3/22/05…$15.00
https://doi.org/10.1145/3524610.3527916

[1] https://pypi.org/

keyword on Twitter, some tweets are returned. However, depending on the context, those tweets either express library sense or other senses.

In recent years, pre-trained Transformer models (PTMs) have brought considerable breakthroughs in Natural Language Processing (NLP) community. Nevertheless, it remains unknown whether PTMs are capable to address the library recognition task well. Given PTMs are generally pre-trained in large corpora, such as Wikipedia and BookCorpus [48], they can learn contextual representations of a word. It makes PTMs a good candidate to solve this unique challenge. They can serve as a baseline for this novel task. In this paper, we examined a broad set of PTMs for the task of library recognition in tweets. We choose five different PTMs - BERT [8], RoBERTa [19], XLNet [43], BERTweet [25], and BERTOverflow [36]. Based on pre-training corpora, they can be roughly divided into two categories: general-purpose, and domain-specific. For each pre-trained model, we fine-tune it on our dataset. We add a linear layer on top of the final hidden state of the `[CLS]` token. We then use the cross-entropy loss function to optimize the model during fine-tuning.

We compare the PTMs to the two closest existing works [28, 45]. Ye et al. [45] present APIReal, a semi-supervised machine learning approach that handles API recognition in Stack Overflow posts. Prasetyo et al. [28] present a framework to identify software-relevant microblogs from the irrelevant ones. Ye et al. [45] deals with API recognition in Stack Overflow posts while our paper deals with library recognition in tweets. Prasetyo et al. [28] identify software-related tweets which could be seen as a broader categorization and a precursor to library recognition, as library-related tweets are always software-related. There are two more recent works on identifying software-related tweets [31, 34], but their approaches rank tweets based on their likelihood to be software-related rather than classifying them. Therefore, their task and ours are fundamentally different. We adopt the two closest existing works to our task of library recognition.

With the lack of a proper dataset, we create a benchmark dataset that contains a total of 4,456 tweets across 23 libraries. We then experiment with three settings, i.e., *within-library*, *cross-library*, and *mixed-library*. In the *within-library* setting, we pick only tweets from a library and split them into training and test sets. In the *cross-library* setting, we use tweets mentioning just one library for testing and the remaining tweets for training. In the *mixed-library* setting, we randomly pick the tweets for training and test. For the mixed-library and within-library settings, we run 5-fold cross-validation, while we run 23 folds for the cross-library setting (as we have a total of 23 libraries).

In most experimental settings, the fine-tuned PTMs outperform the existing approaches that we adapted to the task of library recognition in tweets. For *within-library* setting, RoBERTa achieves the highest average F1-score of 0.84. For *cross-library* setting, RoBERTa and XLNet achieve the highest F1-score of 0.51. For *mixed-library* setting, BERT achieves the highest F1-score of 0.89. They have gained 12%, 9%, and 5% improvement over the best performing baseline, respectively.

Our contributions can be summarized as follows:

- We built a dataset containing tweets that mention 23 libraries as the first benchmark for the task of library recognition in tweets. These 23 libraries are chosen such that they have other commonly-acknowledged senses other than library sense.
- We propose to apply PTMs to library recognition in tweets: disambiguation between library-related tweets and non-library-related tweets.
- We conducted a comprehensive empirical study to evaluate the performance of PTMs in different settings. We also included comparisons related to existing approaches.

Table 1: Example tweets: `Pos` and `Neg` in the Label column indicates that the tweet refers to the library sense of a word or other senses of the word, respectively.

Tweet	Label
"using a python **`flask`** application with an azure ml generated api endpoint https://t.co/o119eu6l28"	Pos
"@sufw water. and the 8oz stanley classic **`flask`** is slim enough to fit into a front pants pocket https://t.co/or4qf2iwky (ttk.me t4ef3)"	Neg
"#protip: "bcrypt-nodejs" is not API compatible with "**`bcrypt`**"... be sure to test your code if you switch from one to the other. #facepalm"	Pos
"a very common hashing algorithm (**`bcrypt`**) has a 72 byte limit though and it's unclear whether pre-hashing might lower security."	Neg

The structure of this paper is as follows. First, we formulate the problem in Section 2. Then we describe the details of our methodology in Section 3. We present our experimental setup in Section 4 including evaluation metrics, existing approaches, the experimental design, and implementations details. We analyze the results of each setting in Section 5. We provide a discussion on the results in Section 6. We present the related work in Section 7. Finally we provide the conclusion of the paper and mention future work in Section 8.

2 PROBLEM FORMULATION

Given a tweet containing a library name, we need to distinguish whether it refers to a library or not. We refer to this problem as the *library recognition task in tweets*. The underlying problem is disambiguation between the library sense and the normal sense of a word. It can be broken down into two subcategories: (i) The words matching the library name is about a library or an entity that is *not* software-related (ii) The words matching the library name is about a library or an entity that is software-related. We show examples in Table 1. To illustrate case (i), consider a Python library name `flask`. Depending on its context, a tweet may refer to the `flask` library or a `flask` object (i.e., a container for liquids), as shown in the first two rows of Table 1. To illustrate case (ii), consider a Python library name `bcrypt`. A tweet containing `bcrypt` may refer to the `bcrypt` library or a password-hashing function, depending on the context of the tweet, as shown in the last two rows of Table 1.

Let us denote a set of tweets as $T = \{t_1, t_2, ..., t_n\}$, each tweet t_k in T contains a library name. We would like to get an answer set $Y = \{y_1, y_2, ..., y_n\}$ where $y_k = 1$ indicates that t_k is related to

a library; $y_k = 0$ means that t_k is not about a library. Therefore, library recognition in tweets is defined as a binary-classification task to predict whether a tweet $t \in T$ contains a library name refers to a library or not, in other words, $F_C = t \rightarrow \{0, 1\}$ such that,

$$F_C(t) = \begin{cases} 1, & \text{if } t \text{ is library-related} \\ 0, & \text{otherwise.} \end{cases} \quad (1)$$

Our aim is to find $F_C(t)$ which can make predictions as accurate as possible.

3 METHODOLOGY

In this paper, we identified a novel task: library recognition in tweets. To benchmark this task, we created the first dataset and evaluated several approaches on this dataset.

3.1 Benchmark Dataset Building

As there have been no dedicated efforts for the library recognition task before, there is a lack of high-quality labeled datasets. Thus, we decide to collect and annotate a dataset well-suited for this task. We made the primary effort to create a Twitter dataset that can be used for library recognition.

Data collection: Firstly, we obtained the 100 most popular Python packages[2] on Dec 21, 2020. We consider these packages as our target libraries. Secondly, we check the library names manually. If they are unique and do not have other commonly-recognized senses, such as `urllib3`, there is no need for further efforts to recognize them. We then have 30 libraries with ambiguous names for experiments. These 30 libraries have normal senses that would need disambiguation from their library sense, e.g., `mock` is a commonly-used English word, `absl` can also be a business term, referring to a conference[3]. Note that the library name and package name are not always the same, for example, to use the library `google protobuf`, the users should install `protobuf` package.[4] When we actually import this library, we should import `google.protobuf` instead. In our data, we focus on library names rather than package names. Therefore, in this example, we consider `google protobuf`.

To get a list of software-relevant tweets, we use the database that we get from a research center[5] that collects tweets from people who are likely to be software developers following the method by Palakorn et al. [4]. These tweets were exclusively collected from the public tweets. In total, the database contains 359,419,640 tweets, ranging from April 2006 to February 2021. From this database, we select tweets that contain all tokens in one of the library names; to cater for minor variations, we ignore the order the tokens appear in a library name. When querying with a library name, a number of tweets were returned. In order to obtain a statistically representative sample of tweets to be further analyzed, we randomly selected a subset of the library's tweets considering a confidence level of 95% and a confidence interval of 5.

Data annotation and cleaning: After getting the tweets, we further clean them by (1) removing all the duplicate tweets; and (2) removing the tweets containing more than one library name from the chosen 30 libraries. (2) is done for two main reasons- (i) when a tweet contains more than one library name, it is extracted each time one of those library names is used as a keyword for extraction, which results in duplicates across different libraries and (ii) each of those library names can be disambiguated to have a library sense or other senses, making it difficult to use the tweet for training or testing, because the label is at the tweet-level instead of the individual token-level. After cleaning, we exclude one library from our dataset since all the tweets mentioning the library were removed. At the end of the process, we have 6,074 tweets, each of which includes the name of one of the 29 libraries.

For the remaining tweets, two annotators labeled them independently. Positive labels indicate that the tweets are actually mentioning a library, while the negative labels indicate that they are not. We excluded the tweets whose labels they disagreed on. The inter-annotator agreement was 0.88, measured by Cohen's Kappa value [21], which indicates almost perfect agreement. There are 6 libraries with zero positive labels, such as `bleach` (i.e., no tweet in our database is referring to the library `bleach`). Therefore, we removed these libraries. In the end, we have a total dataset of 4,456 tweets with 23 library names in our dataset. Table 2 shows the statistics of our dataset.

Table 2: Detailed Statistics of our dataset

Library name	# Pos.	# Neg.	Total
absl	5	11	16
apiclient	6	0	6
appdirs	1	0	1
bcrypt	7	296	303
boto	120	92	212
click	1	379	380
colorama	4	18	22
cryptography	2	352	354
docker	321	41	362
flask	248	93	341
futures	17	349	366
google auth httplib2	1	0	1
google protobuf	41	84	125
ipaddress	5	75	80
mock	62	305	367
more itertools	3	8	11
packaging	2	370	372
pandas	175	178	353
pluggy	13	7	20
prometheus client	7	10	17
prompt toolkit	31	5	36
requests	4	375	379
websocket	13	319	332
Total	**1,089**	**3,367**	**4,456**

[2] https://pythonwheels.com/
[3] https://abslsummit.com/
[4] https://pypi.org/project/protobuf/
[5] https://larc.smu.edu.sg/

3.2 Pre-trained Transformer Models

In NLP field, distributed representations, which are low-dimensional and dense vectors representing the syntactic or semantic features of the language, make it easy to develop various NLP systems [29]. According to Qiu et al. [29], there are two kinds of word embeddings, i.e. non-contextual and contextual word embeddings. The difference between them is whether the embedding of a word dynamically changes based on the context it appears in. Non-contextual embeddings, sometimes are called static word embeddings, and two popular examples are word2vec [22] and GloVe [27]. Contextual embeddings, like BERT [8], can learn contextual representations: the same word will have different representations depending on the context. Therefore, these dynamic embeddings can handle the issue of polysemy (a word can have different senses) and distinguish the semantics of words in different contexts [29]. Since the introduction of the original BERT, many different BERT variants have been proposed in recent years. They are generally pre-trained on a large-scale unlabeled corpus to learn a good representation and then has been shown to be beneficial on many downstream tasks.

In this work, we select BERT [8], RoBERTa [19] and XLNet [43], as they achieved good performance on sentiment classification task for SE [47]. As our task involves SE and tweets, we also considered two domain-specific variants of BERT, i.e., BERTOverflow [36] and BERTweet [25]. We collectively call pre-trained Transformer models as PTMs in the following sections. Based on the pre-training corpus, the PTMs adopted in this work can be divided into two groups: (1) general-purpose: BERT, RoBERTa, and XLNet; and (2) domain-specific: BERTweet, BERTOverflow. As general-purpose PTMs, RoBERTa and XLNet were pre-trained in larger pre-trained corpora than BERT. Among all the PTMs used, the two domain-specific PTMs have the two largest vocabulary sizes. Their details are described as follows (we put the exact model version in the popular Hugging Face library that we use [6] in parentheses):

- **BERT [8]** (`bert-base-uncased`) stands for Bidirectional Encoder Representations from Transformers. It is the first fine-tuning based representation model that achieved state-of-the-art performance on many sentence-level and token-level tasks. BERT is pre-trained with two objective tasks: (1) Masked Language Model (MLM): To train a bidirectional representations, some tokens are masked at random and MLM is aimed to predict these masked tokens. (2) Next Sentence Prediction (NSP): To understand the relationship between two sentences, BERT is also pre-trained with NSP: given two sentences `A` and `B`, predict whether `B` is the sentence after `A`. Two versions of BERT are available, and we follow the original paper to denote the number of layers (i.e., Transformer blocks) as *L*, the hidden size as *H*, and the number of self-attention heads as *A*. These two are $\text{BERT}_{\text{base}}$ (L=12, H=768, A=12, Parameters=110M) and $\text{BERT}_{\text{large}}$ (L=24, H=1024, A=16, Parameters=340M). In this work, we use $\text{BERT}_{\text{base}}$.
- **RoBERTa [19]** (`roberta-base`) is robustly optimized BERT approach and it makes several modifications over BERT. Firstly, it trains the model longer, with bigger batches, over more data. Secondly, it removes the NSP, which is a pre-training task used in BERT. Thirdly, it trains on longer sequences. Finally, it dynamically changes the masking pattern. Other than the BooksCorpus [48] plus English Wikepedia data used by BERT, RoBERTa also included more pre-trained data: CC-News [3] (English portion of the CommonCrawl News dataset); OpenWebText [2] (an open-source recreation of the WebText corpus), and Stories [40] (a subset of CommonCrawl data that match the story-like style). The best RoBERTa model achieved the state-of-the-art performance on several NLP benchmarks. Similar to BERT, RoBERTa also has two versions: $\text{RoBERTa}_{\text{base}}$ (L=12, H=768, A=12, Parameters=125M) and $\text{RoBERTa}_{\text{large}}$ (L=24, H=1024, A=16, Parameters=355M). We also use the base version, i.e., $\text{RoBERTa}_{\text{base}}$.
- **XLNet [43]** (`xlnet-base-cased`) is a generalized autoregressive (AR) method that leverages the advantages of both AR language modeling and autoencoding (AE) while avoiding their limitations. Either AR or AE language modeling has its own pros and cons. For example, AR language modeling seeks to estimate the probability distribution of a text corpus with an AR model. However, an AR language model is only trained to encode a uni-directional context (either forward or backward), which makes it ineffective at modeling deep bidirectional contexts. AE-based models, such as BERT, aim to reconstruct the original data and AE can benefit from the bidirectional contexts. To benefit from both AR and AE, XLNet not only adopts a novel pre-training objective, but also improves architectural designs for pre-training. Compared to BERT, XLNet also involved more pre-training data, such as Common Crawl [1]. XLNet consistently outperforms BERT on a wide spectrum of problems including language understanding tasks, reading comprehension tasks, and text classification tasks. In the Hugging Face library, it also has two variants, i.e., $\text{XLNet}_{\text{base}}$, and $\text{XLNet}_{\text{large}}$, which has the same architecture hyperparameters as $\text{BERT}_{\text{base}}$ and $\text{BERT}_{\text{large}}$, respectively. We adopted the base version of XLNet in this work.
- **BERTweet [25]** (`vinai/bertweet-base`) is the first large-scale pre-trained language model for English tweets. It has the same architecture as $\text{BERT}_{\text{base}}$ and it adopts RoBERTa pre-training procedure. The experimental results demonstrate that BERTweet does better than its competitors, including $\text{RoBERTa}_{\text{base}}$, and outperforms the previous state-of-the-art approaches on three downstream tweet NLP tasks: part-of-speech tagging, named-entity recognition and text classification.
- **BERTOverflow [36]** (`jeniya/BERTOverflow`) is a $\text{BERT}_{\text{base}}$ pre-trained on Stack Overflow 10-year archive of 152 million sentences and 2.3 billion tokens. It is originally designed to help identify code tokens or software-related named entities which appear with natural language sentences, especially on Stack Overflow. The experimental results on the name entity recognition task on Stack Overflow dataset shows that fine-tuning over BERTOverflow improves F1-score by more than 10 points compared to using the off-the-shelf BERT.

3.3 Research Questions

To understand how different approaches perform on this library recognition task, we aim to address the following Research Questions (RQs):

[6] Available on https://huggingface.co/models

RQ1: *How effective are the approaches when trained on tweets from one library in classifying tweets belonging to the same library?*
In this research question, we want to measure how well the approaches work in the scenario where some tweets from a library is known and additional tweets from the same library need to be obtained. We call this setting the *within-library* setting.

RQ2: *How effective are the approaches when trained on tweets from some libraries to classify tweets from another library?*
In this research question, we analyze how the approaches perform in the scenario where we have tweets from a set of libraries and we want to find tweets from another library that are not in the set. We call this setting the *cross-library* setting.

RQ3: *How effective are the approaches in classifying library and non-library related tweets when we mix tweets across various libraries in the training and test sets?*
By answering this question, we want to estimate how well the different approaches are in filtering library and non-library related tweets when we apply it to a random set of tweets that may be related to libraries. We call this setting the *mixed-library* setting.

4 EXPERIMENTAL SETUP

In this section, we explain evaluation metrics, baseline methods, experimental design, and implementation details.

4.1 Evaluation Metrics

As mentioned before, we formulate the library recognition task in tweets as a binary classification task. We apply the following evaluation metrics, i.e., precision, recall, and F1-score, which are commonly used for the binary classification task. The formula to calculate these three metrics are defined as follows:

$$Precision = \frac{TP}{TP + FP} \tag{2}$$

$$Recall = \frac{TP}{TP + FN} \tag{3}$$

$$F1 = \frac{2 \times (Precision \times Recall)}{Precision + Recall} \tag{4}$$

TP, FP, FN represents true positives, false positives, and false negatives, respectively. TP indicates the number of tweets that mention a library (i.e., they have a positive label) and have been classified as positive correctly. Similarly, FP is the number of the tweets that do not mention a library (i.e., they have a negative label) and have been mistakenly classified as positive. FN is the number of tweets that mention a library (i.e., they have a positive label) and are mistakenly classified as negative. $F1$ is a weighted average of the $Precision$ and $Recall$, where $F1$ ranges from 0 to 1; the larger, the better. In our work, we use F1-score, which measures the trade-off between precision and recall, as the main evaluation metric.

4.2 Methods to Compare With

Although there is no existing approach that is specially designed for library recognition in tweets, we identified two closest pieces of work that can serve as the baselines. Therefore, we adapt them for our task to understand how fine-tuned PTMs perform in the task of library recognition in tweets. We use the same experimental design settings and evaluation metrics outlined later for each approach.

- **Strawman:** All the tweets that we have collected for our dataset contain library names as keywords. In this approach, we assume that all of these tweets in fact refer to library names and mark all the tweets as positive. We refer to this as the *Strawman* approach.
- **APIReal [45] :** APIReal is used for API recognition and linking in Stack Overflow posts. API and library are close concepts in SE, thus, we regard API recognition as similar to library recognition. Only the first part of APIReal, i.e. API recognition, has been leveraged and implemented as it is the only part that is relevant to our task of library recognition in tweets. The steps of API recognition consist of: (1) text pre-processing steps including code snippet removal, HTML tag clearing, and tokenization, (2) usage of two unsupervised language models (i.e., class-based Brown clustering [5] and neural-network-based word embedding [22, 41]) to learn word representations from unlabeled text and cluster semantically similar words, (3) constructing an API inventory containing the generated clusters, (4) building a linear-chain Conditional Random Field (CRF) model [35] using orthographic features from tokens, compound word-representation features from the two different unsupervised language models, and gazetteer feature from the API inventory and (5) performing an iterative self-training process to alleviate the lack of labeled data for model training. The final trained model would be used to check if a sentence contains mentions of API or not. Note that APIReal is a token-based approach and it recognizes tokens that may refer to public modules, classes, methods, or functions of certain libraries as API mentions. We adjust the model to our task by labeling the library names as "APIs". The model would then learn whether each token is library-related. As our task is tweet-based, in the inference stage, if any of the tokens are recognized as positive, we regard the whole tweet is recognized as positive. For library names with more than one token, we labeled all the tokens in the library name as positive. We included the original replication package by APIReal [7] and the adaptation scripts we wrote in our replication package.
- **Prasetyo et al. [28]** : Prasetyo et al. proposed a framework to identify the relevant software micro-blogs from the irrelevant ones. As a tweet mentioning a library is certainly a software-related one, but not vice versa, we consider their task is a broader one over our library recognition task. Their framework consists of three components: (1) a webpage crawler, (2) a text processor, and (3) a machine learning classifier. A set of features are extracted from the tweet content as well as their embedded URLs. Then they train a discriminative model using Support Vector Machine (SVM) as the classifier. This discriminative model is then used to classify the tweets as software-relevant or not. We re-trained this approach on our dataset to make it able to conduct library recognition on tweets.

4.3 Experimental Design

We run extensive experiments to answer each RQ. Inspired by settings consider by prior works (on defect prediction) [16, 37, 46], we first consider *within-library* and *cross-library* settings in RQ1 and RQ2. Other than these two settings, we further conducted an extra setting that mixed libraries.

[7] https://github.com/baolingfeng/APIExing

- **Within-library:** In this setting, we train the model on the tweets mentioning a library name and test on the tweets containing the same library name. As many libraries in our dataset only contains a small number of tweets and a limited number of positive samples, we can only evaluate the within-library setting on five libraries: `boto`, `docker`, `flask`, `mock` and `pandas`. Thus, for each of these five libraries in our dataset, we conduct a 5-fold cross-validation. At each fold, 80% of the data is used for training, and the remaining 20% is used for testing. We report the averaged precision, recall, and F1-score across the 5 folds.
- **Cross-library:** In this setting, we train and test the model with tweets containing different library names. As mentioned earlier, our final dataset consists of a total of 23 libraries. In this setting, we use the tweets in 22 libraries to train the model, and the tweets containing the remaining library name will be used for testing. Therefore, each library is used as a test dataset once. We report the average precision, recall, and F1-score across 23 runs.
- **Mixed-library:** In this setting, we do not distinguish between different library names. The training and test set may contain tweets with the same or different library names. We run five-fold cross-validation on the entire tweets. At each fold, 80% of the tweets is used for training, and the remaining 20% is used for testing. We report the average precision, recall, and F1-score across the 5 folds as the final result.

4.4 Implementation Details

In this part, we detailed how we implemented PTMs for our task as follows:

Pre-processing: Other than common words, library names are often mentioned with `@` (i.e., entity mentions) or `#` (i.e., hashtags). We keep both special tokens. We pre-processed all the tweets as follows: (1) lower-casing all the words; (2) removing URLs, retweet abbreviation ('rt'); and (3) removing all trailing white spaces.

Fine-tuning Pre-trained Transformer Models: We fine-tuned the PTMs to solve the library recognition task on tweets. Figure 1 illustrates the overall architecture of the fine-tuned PTMs. The input is a single tweet, and the output is the predicted class label: library-related (positive) or not library-related (negative). According to the original paper [8], the first output token of every sequence is always a special classification token (`[CLS]`). For classification tasks, we can use the final hidden state of this token as the aggregate sequence representation. Therefore, for each PTM, we add a fully-connected linear layer on top of the final hidden state of the `[CLS]` token. We then apply the sigmoid function to scale the output from the linear layer to obtain the probability of the predicted label. We train the models with the cross-entropy loss and use AdamW [20] with a linear learning rate scheduler for optimization. The formula to calculate the cross-entropy loss is as follows:

$$\mathcal{L}(\boldsymbol{y}, \hat{\boldsymbol{y}}) = -y_1 \cdot \log(\hat{y}_1) - (1 - y_1) \cdot \log(1 - \hat{y}_1) \quad (5)$$

where y_1 represents the probability of positive class in ground truth and $\hat{y_1}$ represents the predicted probability of positive class. We implemented PTMs for classification with PyTorch [26] and Hugging Face library [42].

In case the model overfits the training data, we use 10% of the training data for validation. We run a model for 10 epochs, so for each epoch, we get a checkpoint of the current model. As our main evaluation metric is F1-score, we save the model checkpoint that achieves the highest F1-score on the validation set across the 10 epochs. In the inference stage, we load these best checkpoints to predict the labels for tweets on the held-out test data. The hyper-parameters used in our experiments are: (1) Learning rate: $2e-5$; (2) Batch size: 32; (3) Max length: 100; (4) Epochs: 10. We run all the experiments on a Linux machine that is equipment with GeForce RTX 2080Ti.

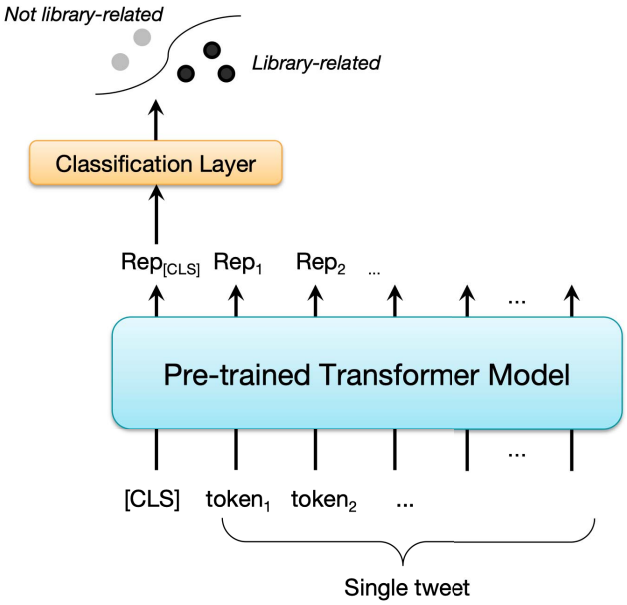

Figure 1: The architecture of fine-tuned PTMs

Table 3: Within-library: 5-fold averaged Precision, Recall, F1-score

Approach	Precision	Recall	F1
Strawman	0.59	1.0	0.69
APIReal	0.79	0.75	**0.75**
Prasetyo et al. [28]	0.63	0.72	0.66
PTM			
– BERT	0.82	0.83	0.81
– **RoBERTa**	0.83	0.87	**0.84**
– XLNet	0.76	0.79	0.77
– BERTweet	0.84	0.82	0.81
– BERTOverflow	0.6	0.75	0.66

5 RESULTS

In this section, we demonstrate the details of our results to answer the RQs.

Table 4: Within-library: Detailed F1-scores of all the approaches in each library

Approach	`boto`	`docker`	`flask`	`mock`	`pandas`
Strawman	0.72	0.94	0.84	0.29	0.66
APIReal	0.92	0.93	0.88	0.16	0.84
Prasetyo et al. [28]	0.73	0.94	0.85	0	0.76
PTM					
– BERT	0.96	0.94	0.96	0.3	0.91
– RoBERTa	0.96	0.94	0.95	0.44	0.92
– XLNet	0.94	0.94	0.96	0.09	0.91
– BERTweet	0.95	0.94	0.96	0.29	0.93
– BERTOverflow	0.84	0.94	0.84	0	0.69

5.1 Effectiveness in the Within-Library Setting

Table 3 demonstrates the overall results on the within-library setting. It shows the average metrics for 5-fold cross validation performed on each of the 5 selected libraries - `boto`, `docker`, `flask`, `mock` and `pandas`. RoBERTa has the highest F1-score among the PTMs, while BERTOverflow has the lowest F1-score. Regarding F1-score, RoBERTa beats APIReal by 12%, Strawman by 22%, and Prasetyo et al. [28] by 27%. The PTMs have high scores with precision ranging from 0.6 to 0.82, recall ranging from 0.75 to 0.87, and F1-score ranging from 0.66 to 0.84. All the PTMs achieve high scores for all libraries except `mock`. In this setting, APIReal's performance is lower than that of the other four PTMs, while it is better than BERTOverflow.

Table 4 shows the detailed F1-score produced by each approach for each library. By looking at Tables 3 and 4 together, we observe that the performance of models for different libraries closely follows the positive and negative label distribution of the tweets. Specifically, the performance is high if the number of positive tweets in the dataset is nearly equal to or more than the number of negative tweets used for training. In other words, the performance is high if the dataset is mostly balanced. In the case of `mock`, when the number of positive tweets is significantly less, the performances of models are also less, with BERTOverflow having the lowest F1-score of 0. Another interesting result is BERTOverflow having a high F1-score of 0.94 for `docker` while it ranges from 0 to 0.84 for the remaining libraries. This is expected considering that `docker` is a widely used software terminology and `mock` is not a SE-related word. Moreover, BERTOverflow is exclusively pre-trained on software-related data from Stack Overflow posts. From the experiment results, we can conclude that in a within-library setting, fine-tuned PTM performs better than the baseline approaches.

> **Within-library setting:** The fine-tuned PTMs outperform all the baseline approaches, with the exception of BERTOverflow. The best-performing PTM (RoBERTa) get an averaged F1-score of 0.84, which beats the best performing baseline - APIReal by 12%.

Table 5: Cross-library: averaged Precision, Recall, F1-score - for each approach, we run it 23 times, each time, 22 libraries for training, and the rest library is for testing.

Approach	Precision	Recall	F1
Strawman	0.39	1.0	**0.47**
APIReal	0.22	0.07	0.08
Prasetyo et al. [28]	0.66	0.38	0.41
PTM			
– BERT	0.59	0.65	0.47
– **RoBERTa**	0.6	0.63	**0.51**
– **XLNet**	0.62	0.65	**0.51**
– BERTweet	0.53	0.62	0.48
– BERTOverflow	0.51	0.49	0.4

5.2 Effectiveness in the Cross-Library Setting

Table 5 demonstrates the results on the cross-library setting. For each approach, the metrics are averaged across 23 folds. This setting has the lowest performance across the three settings. Simply treating all the tweets as library-related (Strawman) approach produced higher F1-scores than the other two baseline approaches, APIReal and Prasetyo et al. [28]. It indicates that the current approaches are not good at recognizing libraries in tweets under the cross-library setting. Especially, APIReal has the lowest performance here with an averaged F1-score of only 0.08. This may suggest that APIReal does not have a good ability to transfer the knowledge it learns from some libraries to another given its limited knowledge base.

All the PTMs have produced precision ranging from 0.51 to 0.62, recall ranging from 0.49 to 0.65, and F1-score ranging from 0.4 to 0.51. In this setting, APIReal's performance is much lower than that of the PTMs. Prasetyo et al. [28] got much higher F1-score over APIReal. Here, RoBERTa and XLNet have the best performance and beat Prasetyo et al. [28] by 24% and Strawman by 9%. BERT and BERTweet also have good performance with almost the same F1-score. BERTOverflow has the lowest performance among the PTMs, lower by at least 0.06 from the second-lowest model (BERT). Although all the approaches perform worse in the cross-library setting compared to them in the within-library setting, PTMs are still able to produce a better result than the rest approaches. It can be concluded that in the cross-library setting, PTMs are better than other existing approaches. Specifically, fine-tuned RoBERTa and XLNet perform better than the other PTMs and existing approaches.

> **Cross-library setting:** All of the fine-tuned PTMs outperform the existing approaches. RoBERTa and XLNet have the best performance among the PTMs. They improved the F1-score over the best baseline approach - Strawman by 9%.

5.3 Effectiveness in the Mixed-Library Setting

Table 6 demonstrates the results on the mixed-library setting. It shows the average metrics for 5-fold cross-validation on all 23

Table 6: Mixed-library: 5-fold averaged Precision, Recall, F1-score

Approach	Precision	Recall	F1
Strawman	0.24	1.0	0.39
APIReal	0.88	0.83	**0.85**
Prasetyo et al. [28]	0.92	0.59	0.72
PTM			
– **BERT**	0.89	0.89	**0.89**
– RoBERTa	0.87	0.88	0.87
– XLNet	0.85	0.89	0.87
– BERTweet	0.86	0.9	0.88
– BERTOverflow	0.85	0.78	0.81

libraries. APIReal and Prasetyo et al. [28] can achieve better performance than Strawman by a large margin. It indicates that directly treating all the tweets with a library name as library-related is far from satisfactory. All the PTMs have high scores with precision ranging from 0.85 to 0.89, recall ranging from 0.78 to 0.9, and F1-score ranging from 0.81 to 0.89. In this setting, APIReal's performance is close to that of the PTMs, and it is significantly better than Prasetyo et al. [28]. Fine-tuned BERT has the highest F1-score and beats Prasetyo et al. [28] by 24% and APIReal by 5%. BERTOverflow has the lowest performance among the PTMs by being at least 0.06 less than the other PTMs. Comparing the results from the within-, cross-, and mixed-library setting, it can be observed that in the mixed-library setting, all the approaches can achieve higher F1-scores. Consider that in the mixed-library setting, the training data is more than that in the within-library setting, and the predicted libraries may also be included in the training data. Inference in the mixed-library setting can leverage more information than the other two settings. It can be concluded that in the mixed-library setting, fine-tuned BERT performs better than the other PTMs and existing approaches.

> **Mixed-library setting:** The fine-tuned PTMs can outperform all the existing approaches in this setting, with the exception of BERTOverflow. The best-performing PTM (BERT) get an averaged F1-score of 0.89, which beats the best performing baseline - APIReal by 5%.

6 DISCUSSION

6.1 Learning from Multiple Sources of Knowledge

Fine-tuned PTMs are able to perform better than the existing models when conducting the library recognition task in tweets. It indicates that fine-tuning PTMs can serve as a strong baseline for this new task. Furthermore, we find that different settings produce varying results across the different PTMs, c.f., Fig. 2 for a visual comparison of the F1-scores of different models in different settings. On average, PTMs perform the best in the mixed-library setting followed by the within-library setting and the cross-library setting. It is expected

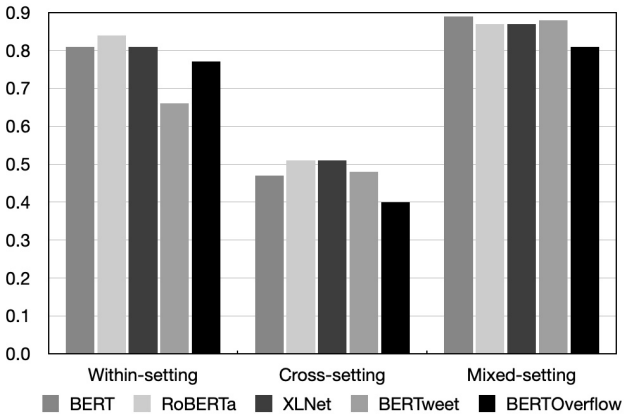

Figure 2: Comparison of F1-scores of the PTMs in different settings

Table 7: Example misclassified tweets by BERT in mix-library setting

Tweet
"Keen to master #Python's best development practices and module system? Take a look at @robsmallshire hands on course! For more info **`click`** here: https://bit.ly/2mkVa5v"
"@bphogan It's frustrating that **`packaging`** Ruby tools up isn't even worth the time. Dropbox was built with alot of Python, but did we notice?"
"@gnuconsulting @zobar2 FanExpo! I got a fun **`flask`**"
"@gegere @fredjean #**`docker`** and coffee"

that the PTMs perform better for the within-library setting as compared to the cross-library setting. In the within-library setting, the learned knowledge is within the same domain, as the PTMs are used to classify a library using the knowledge about the library itself. In the cross-library setting, PTMs need to transfer the knowledge that they learn from some libraries to classify a totally different one. According to the results, some knowledge is apparently useful but some is not. The interesting part is that PTMs perform the best in the mixed-library setting, which can be viewed as a mixture of both the within-library and the cross-library settings. This suggests that we should learn knowledge from the same library as well as leverage knowledge from other libraries to boost the performance of PTMs.

6.2 Analyzing the Misclassified Tweets by PTMs

To understand the failed cases produced by PTMs, we take a deep analysis of the fine-tuned BERT under the within-library setting. Table 7 shows some example tweets. We observe several emerging patterns like follows:

(i) The tweet is software-related but the library name does not refer to its library sense. For example, consider the first two rows in Table 7. The first tweet contains the keyword `click`. Since `click` here refers to a verb, the tweet was marked as negative (not library-related). However, BERT classifies it as a relevant tweet, perhaps since the tweet is heavy with terms relevant to software development. It also talks about the module system, which is closely related to the library. The second tweet was extracted with keyword `packaging`. This tweet also has a software context but `packaging` is the "-ing" form of `package` and it is used as a gerund, which has nothing to do with the `packaging` library. However, the models cannot capture this information and classify the tweet as relevant.
(ii) Very short tweets. Short tweets with no meaningful context are marked as irrelevant, but BERT marks them as relevant in some cases. The last two tweets from Table 7 are both quite short with limited information. For the first tweet, it is simple for us to decide that this `flask` is not the Python library. For the second one, we cannot conclude that it is library-related given the insufficient context. It probably refers to the general docker software. Both tweets have negative labels in our data, however, BERT predicted them are positive. Moreover, the second tweet actually contains the image after the short text in the original tweet link. Leveraging visual information is a possible direction to explore in the future.

From these misclassified examples, we can see that library recognition in tweets is not a trivial task, especially with the limited number of words in a tweet.

6.3 Need of a Specialized Approach

Although fine-tuning PTMs is promising in this novel task - library recognition in tweets – we believe that more efforts should be made. Considering the generally poor performance in a cross-library setting, a detailed and specialized approach is required to achieve satisfactory performance. Given that tweets are generally short, PTMs are likely the way to go. The PTMs can learn knowledge from other contexts, and transfer the knowledge to library recognition. The PTMs utilized in this work are pre-trained general knowledge, like Wikipedia, or domain-specific knowledge, such as Stack Overflow posts. None of them perfectly contain the knowledge to address the library recognition in tweets. A potentially better pre-training corpus is a dataset of tweets containing software-related knowledge. The benefits are twofold, they are from the same platform and the context is similar. We encourage future work to design a domain-specific PTM for library recognition in tweets.

6.4 Threats to Validity

Threats to internal validity relate to the experiment bias. Our ground-truth dataset is manually labeled. Like other manual labeling activities, this procedure can be biased or inconsistent. To alleviate this threat, two annotators were involved and they both have at least 4-year experience in computer programming. They labeled the data independently. We have achieved the inter-annotator agreement of 96.3%, and the Cohen's Kappa value [21] is 0.88, which indicates almost perfect agreement. We then discarded the tweets they disagreed with. Therefore, we only experimented on the tweets both of them agreed on. We include the labeled dataset in our replication package for others to inspect.

Threats to external validity relate to our findings may not be generalized to other library names. In this work, we have investigated the 100 most popular Python libraries and randomly chose 30 libraries for our experiments – we retain 23 of them, as for the other 7, we do not have any tweet about the library from sampled tweets from our database. Since we only experimented with Python library names, it is unclear whether similar results can be achieved for libraries in other languages. We pick Python as it is one of the most popular programming languages today.

Threats to construct validity relate to the evaluation metrics used in our experiments. As our task is a binary text classification task, we followed the prior work [34, 45, 47] to use precision, recall, and F1-score to evaluate the model performance. For all the three experimental settings in our work, we reported the averaged results. Specifically, for mixed- and within-library settings, we run 5-fold cross-validation. For the cross-library setting, each library has been used for the test set once. And we hereby report the 23-fold averaged performance. Thus, we believe the threats are minimal.

7 RELATED WORK

Identifying Software Engineering Related Tweets. Identifying SE-related tweets can be seen as a broader task. A number of studies have focused on this topic (e.g., [28, 34]). We already introduced Prasetyo et al. [28] in Section 4.2. Sulistya et al.[34] investigated adopting the knowledge from SE Stack Exchange and Stack Overflow to automate knowledge-seeking tasks in other less software-development-specific platforms, i.e., Twitter and YouTube. One of the two tasks they investigated is identifying software-related tweets. Our work is different from the above work as we focus on a more fine-grained classification task, i.e., library recognition task: we disambiguate whether a tweet that contains a library name actually refers to its library sense or normal sense. Moreover, not all of the aforementioned work cast the problem as a classification task. For example, Sulistya et al. cast it as a ranking problem.

Disambiguation in NLP and SE. Our work can be categorized as disambiguation in SE, as we disambiguate the library sense of a common word from its other senses. Our work is close to word sense disambiguation (WSD) [24], which is a well-researched topic in NLP. WSD determines the sense of polysemous words in a given context. The difference between our work and classic WSD is that we do not distinguish among many commonly-used senses for a word. We only distinguish between library sense and non-library sense. In other words, our focus is whether we can identify the library sense accurately. The WSD itself is a challenging problem and it is out of the scope of our paper.

Several efforts have been devoted to creating software-specific word similarity databases. Although they do not directly deal with WSD, they highlight the different word senses in the software context [38, 39]. Tian et al. observed that software-specific words are generally not contained in WordNet [23], and the meanings of the same words in WordNet are often different than when they are used in the SE context. They [38] propose a technique to automatically construct a database called $WordSim^{SE}_{DB}$ that stores the similarity of words used in the SE context. They [39] also presented a word-similarity database for the SE community: SEWordSim, which is

trained from software-specific documents. Their evaluation shows that considering software context allows SEWordSim to be more accurate than WordNet-based approaches on words related to software development.

Recently, disambiguation in SE has gained more attention [9, 45]. One particular application is API recognition. Ye et al. [45] proposed a semi-supervised machine learning approach for API recognition in informal text, specifically, Stack Overflow text. We have discussed their method in Section 4.2. They claimed that API recognition is the prerequisite to API linking, which aims to link the recognized API to its unique fully qualified name. Other than the API linking problem, API recognition can benefit other tasks, such as API recommendation [30], and bug fixing [7]. Similarly, we believe the library recognition task tackled in this paper can also benefit many other tasks, such as library recommendation. Another application is disambiguating software requirements. Ezzini et al. [9] focus on handling coordination ambiguity and prepositional-phrase attachment ambiguity due to the prevalence of these types of ambiguity in natural-language requirements. Their approach is based on the automatic generation of a domain-specific corpus from Wikipedia. Their evaluation also shows that integrating domain knowledge can lead to an improvement in the accuracy of ambiguity detection and interpretation. Different from the aforementioned two works, our work focuses on disambiguating the library sense of a word from the normal sense of the word in tweets. Our work and these works are complementary. Together, we aim to solve ambiguity issues in SE, while concentrating on different aspects.

Pre-trained Transformer Models for Software Engineering. Given the success witnessed in many NLP tasks, researchers in the SE field have started to utilize PTMs to solve many SE-related tasks as well. There are generally two lines of research of PTMs for SE. The first focuses on pre-training large-scale domain-specific models in SE field [11, 12, 17]. One example is BERTOverflow [36], which has been introduced in Section 3. Another example is CodeBERT [11], which is a bimodal pre-trained model for both programming languages and natural languages. The experimental result showed that fine-tuning CodeBERT achieved state-of-the-art performance on two downstream tasks, i.e., natural language code search and code-to-documentation generation. Similar to CodeBERT, there are also many PTMs that have been pre-trained on programming languages, such as GraphCodeBERT [12] considers the inherent structure of code.

The second one pays attention to fine-tuning PTMs for boosting the performance in downstream SE tasks [18, 47]. For example, traceability recovery [18], sentiment analysis for software artifacts [47] and code search [11]. Zhang et al. [47] conducted an empirical study, which compared the four general-purpose PTMs with four widely-adopted sentiment analysis tools in SE. The experimental results demonstrate the superior performance of PTM on sentiment analysis for SE. It suggested that PTMs are more ready for practice use than prior existing sentiment analysis tools, such as Senti4SD [6]. Lin et al. [18] utilized CodeBERT, a variant of BERT which was pre-trained on source code and documents, to recover links between issues and commits on open-source projects. The results indicate that the architecture that applies CodeBERT generates trace links more accurately than the state-of-the-art approaches, e.g., RNN. Both these two works demonstrate that PTMs can benefit the downstream tasks. Similar to these two works, our work also exploits PTMs to boost the performance for SE. Different from them, we focus on the library recognition task and we also adopted domain-specific PTMs, i.e., BERTweet and BERTOverflow.

8 CONCLUSION AND FUTURE WORK

In this work, we focus on addressing the library recognition challenge in tweets, i.e., disambiguation between the library sense of a common word and the normal sense of the word in tweets. We built the first benchmark dataset for this task. As a starting point, we also conducted extensive experiments which utilized the powerful PTMs to solve this challenge. Specifically, we worked with pre-trained BERT, RoBERTa, XLNet, BERTweet, and BERTOverflow for this study and compare their performances with three baselines.

Our study shows that the fine-tuned PTMs are able to produce better results than the existing approaches. In all of the experimental settings, at least one of the fine-tuned PTMs outperforms the existing approaches that we adapted to the library recognition task. The best models across different settings are different. Different approaches also produce varying results across different settings. For *within-library* setting, RoBERTa gets the highest average F1-score of 0.84. For *cross-library* setting, RoBERTa and XLNet get the highest F1-score of 0.51. For *mixed-library* setting, BERT has the highest F1-score of 0.89. The models show different performances for different settings and show some fluctuations within a setting based on the dataset. Fine-tuned PTMs are effectively able to classify tweets as being related to a library or not.

In the future, we would like to evaluate PTMs for the library recognition task using libraries written in other programming languages, such as Java. We are also interested in designing a specific solution to recognize the library more accurately. Furthermore, we would like to adopt the library recognition result for more downstream tasks, such as the library recommendation.

Replication Package. We make the replication package of our work available online[8].

ACKNOWLEDGMENT

This research / project is 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] [n.d.]. Common Crawl. http://commoncrawl.org/. (Accessed on 10/22/2021).
[2] [n.d.]. Download | OpenWebTextCorpus. https://skylion007.github.io/OpenWebTextCorpus/. (Accessed on 10/22/2021).
[3] [n.d.]. News Dataset Available – Common Crawl. https://commoncrawl.org/2016/10/news-dataset-available/. (Accessed on 10/22/2021).
[4] Palakorn Achananuparp, Ibrahim Nelman Lubis, Yuan Tian, David Lo, and Ee-Peng Lim. 2012. Observatory of trends in software related microblogs. In *2012 Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering*. 334–337. https://doi.org/10.1145/2351676.2351740

[8] https://github.com/soarsmu/Library-Recognition-In-Tweets

[5] Peter F Brown, Vincent J Della Pietra, Peter V Desouza, Jennifer C Lai, and Robert L Mercer. 1992. Class-based n-gram models of natural language. *Computational linguistics* 18, 4 (1992), 467–480.

[6] Fabio Calefato, Filippo Lanubile, Federico Maiorano, and Nicole Novielli. 2018. Sentiment polarity detection for software development. *Empirical Software Engineering* 23, 3 (2018), 1352–1382.

[7] Fuxiang Chen and Sunghun Kim. 2015. Crowd debugging. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*. 320–332.

[8] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* (2018).

[9] Saad Ezzini, Sallam Abualhaija, Chetan Arora, Mehrdad Sabetzadeh, and Lionel C Briand. 2021. Using domain-specific corpora for improved handling of ambiguity in requirements. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 1485–1497.

[10] Hongbo Fang, Daniel Klug, Hemank Lamba, James Herbsleb, and Bogdan Vasilescu. 2020. Need for tweet: How open source developers talk about their github work on twitter. In *Proceedings of the 17th International Conference on Mining Software Repositories*. 322–326.

[11] Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, et al. 2020. Codebert: A pre-trained model for programming and natural languages. *arXiv preprint arXiv:2002.08155* (2020).

[12] Daya Guo, Shuo Ren, Shuai Lu, Zhangyin Feng, Duyu Tang, Shujie Liu, Long Zhou, Nan Duan, Alexey Svyatkovskiy, Shengyu Fu, et al. 2020. Graphcodebert: Pre-training code representations with data flow. *arXiv preprint arXiv:2009.08366* (2020).

[13] Emitza Guzman, Rana Alkadhi, and Norbert Seyff. 2016. A needle in a haystack: What do twitter users say about software?. In *2016 IEEE 24th international requirements engineering conference (RE)*. IEEE, 96–105.

[14] Emitza Guzman, Rana Alkadhi, and Norbert Seyff. 2017. An exploratory study of Twitter messages about software applications. *Requirements Engineering* 22, 3 (2017), 387–412.

[15] Emitza Guzman, Mohamed Ibrahim, and Martin Glinz. 2017. A little bird told me: Mining tweets for requirements and software evolution. In *2017 IEEE 25th International Requirements Engineering Conference (RE)*. IEEE, 11–20.

[16] Xiao-Yuan Jing, Fei Wu, Xiwei Dong, and Baowen Xu. 2016. An improved SDA based defect prediction framework for both within-project and cross-project class-imbalance problems. *IEEE Transactions on Software Engineering* 43, 4 (2016), 321–339.

[17] Aditya Kanade, Petros Maniatis, Gogul Balakrishnan, and Kensen Shi. 2020. Learning and evaluating contextual embedding of source code. In *International Conference on Machine Learning*. PMLR, 5110–5121.

[18] Jinfeng Lin, Yalin Liu, Qingkai Zeng, Meng Jiang, and Jane Cleland-Huang. 2021. Traceability transformed: Generating more accurate links with pre-trained BERT models. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 324–335.

[19] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692* (2019).

[20] Ilya Loshchilov and Frank Hutter. 2017. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101* (2017).

[21] Mary L McHugh. 2012. Interrater reliability: the kappa statistic. *Biochemia medica* 22, 3 (2012), 276–282.

[22] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*. 3111–3119.

[23] George A Miller. 1995. WordNet: a lexical database for English. *Commun. ACM* 38, 11 (1995), 39–41.

[24] Roberto Navigli. 2009. Word sense disambiguation: A survey. *ACM computing surveys (CSUR)* 41, 2 (2009), 1–69.

[25] Dat Quoc Nguyen, Thanh Vu, and Anh Tuan Nguyen. 2020. BERTweet: A pre-trained language model for English Tweets. *arXiv preprint arXiv:2005.10200* (2020).

[26] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (Eds.), Vol. 32. Curran Associates, Inc. https://proceedings.neurips.cc/paper/2019/file/bdbca288fee7f92f2bfa9f7012727740-Paper.pdf

[27] Jeffrey Pennington, Richard Socher, and Christopher D Manning. 2014. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*. 1532–1543.

[28] Philips K Prasetyo, David Lo, Palakorn Achananuparp, Yuan Tian, and Ee-Peng Lim. 2012. Automatic classification of software related microblogs. In *2012 28th IEEE International Conference on Software Maintenance (ICSM)*. IEEE, 596–599.

[29] Xipeng Qiu, Tianxiang Sun, Yige Xu, Yunfan Shao, Ning Dai, and Xuanjing Huang. 2020. Pre-trained models for natural language processing: A survey. *Science China Technological Sciences* (2020), 1–26.

[30] Mohammad Masudur Rahman, Chanchal K Roy, and David Lo. 2016. Rack: Automatic api recommendation using crowdsourced knowledge. In *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, Vol. 1. IEEE, 349–359.

[31] Abhishek Sharma, Yuan Tian, and David Lo. 2015. NIRMAL: Automatic identification of software relevant tweets leveraging language model. In *2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*. IEEE, 449–458.

[32] Leif Singer, Fernando Figueira Filho, and Margaret-Anne Storey. 2014. Software engineering at the speed of light: how developers stay current using twitter. In *Proceedings of the 36th International Conference on Software Engineering*. 211–221.

[33] Margaret-Anne Storey, Christoph Treude, Arie van Deursen, and Li-Te Cheng. 2010. The impact of social media on software engineering practices and tools. In *Proceedings of the FSE/SDP workshop on Future of software engineering research*. 359–364.

[34] Agus Sulistya, Gede Artha Azriadi Prana, Abhishek Sharma, David Lo, and Christoph Treude. 2020. Sieve: Helping developers sift wheat from chaff via cross-platform analysis. *Empirical Software Engineering* 25, 1 (2020), 996–1030.

[35] Charles Sutton, Andrew McCallum, and Khashayar Rohanimanesh. 2007. Dynamic Conditional Random Fields: Factorized Probabilistic Models for Labeling and Segmenting Sequence Data. *J. Mach. Learn. Res.* 8 (2007), 693–723. http://dl.acm.org/citation.cfm?id=1314523

[36] Jeniya Tabassum, Mounica Maddela, Wei Xu, and Alan Ritter. 2020. Code and named entity recognition in stackoverflow. *arXiv preprint arXiv:2005.01634* (2020).

[37] Sadia Tabassum, Leandro L Minku, Danyi Feng, George G Cabral, and Liyan Song. 2020. An investigation of cross-project learning in online just-in-time software defect prediction. In *2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE)*. IEEE, 554–565.

[38] Yuan Tian, David Lo, and Julia Lawall. 2014. Automated construction of a software-specific word similarity database. In *2014 Software Evolution Week-IEEE Conference on Software Maintenance, Reengineering, and Reverse Engineering (CSMR-WCRE)*. IEEE, 44–53.

[39] Yuan Tian, David Lo, and Julia Lawall. 2014. SEWordSim: Software-specific word similarity database. In *Companion Proceedings of the 36th International Conference on Software Engineering*. 568–571.

[40] Trieu H Trinh and Quoc V Le. 2018. A simple method for commonsense reasoning. *arXiv preprint arXiv:1806.02847* (2018).

[41] Joseph Turian, Lev Ratinov, and Yoshua Bengio. 2010. Word representations: a simple and general method for semi-supervised learning. In *Proceedings of the 48th annual meeting of the association for computational linguistics*. 384–394.

[42] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Remi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander Rush. 2020. Transformers: State-of-the-Art Natural Language Processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. Association for Computational Linguistics, Online, 38–45. https://doi.org/10.18653/v1/2020.emnlp-demos.6

[43] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Russ R Salakhutdinov, and Quoc V Le. 2019. Xlnet: Generalized autoregressive pretraining for language understanding. *Advances in neural information processing systems* 32 (2019).

[44] Muhammad Yasir, Kevin Michael, Bastin Tony Roy Savarimuthu, and Sherlock A Licorish. 2018. Formal in the Informal: A Multi-Level Analysis of Core Python Developers' Tweets. In *2018 25th Australasian Software Engineering Conference (ASWEC)*. IEEE, 151–160.

[45] Deheng Ye, Lingfeng Bao, Zhenchang Xing, and Shang-Wei Lin. 2018. APIReal: an API recognition and linking approach for online developer forums. *Empirical Software Engineering* 23, 6 (2018), 3129–3160.

[46] Zhengran Zeng, Yuqun Zhang, Haotian Zhang, and Lingming Zhang. 2021. Deep just-in-time defect prediction: how far are we?. In *Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis*. 427–438.

[47] Ting Zhang, Bowen Xu, Ferdian Thung, Stefanus Agus Haryono, David Lo, and Lingxiao Jiang. 2020. Sentiment Analysis for Software Engineering: How Far Can Pre-trained Transformer Models Go?. In *2020 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 70–80.

[48] Yukun Zhu, Ryan Kiros, Rich Zemel, Ruslan Salakhutdinov, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. 2015. Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. In *Proceedings of the IEEE international conference on computer vision*. 19–27.