ABSTRACT

Stack Overflow is often viewed as one of the most influential Software Question & Answer (SQA) websites, containing millions of programming-related questions and answers. Tags play a critical role in efficiently structuring the contents in Stack Overflow and are vital to support a range of site operations, e.g., querying relevant contents. Poorly selected tags often introduce extra noise and redundancy, which raises problems like tag synonym and tag explosion. Thus, an automated tag recommendation technique that can accurately recommend high-quality tags is desired to alleviate the problems mentioned above.

Inspired by the recent success of pre-trained language models (PTMs) in natural language processing (NLP), we present PTM4Tag, a tag recommendation framework for Stack Overflow posts that utilize PTMs with a triplet architecture, which models the components of a post, i.e., Title, Description, and Code with independent language models. To the best of our knowledge, this is the first work that leverages PTMs in the tag recommendation task of SQA sites. We comparatively evaluate the performance of PTM4Tag based on five popular pre-trained models: BERT, RoBERTa, ALBERT, CodeBERT, and BERTOverFlow. Our results show that leveraging CodeBERT, a software engineering (SE) domain-specific PTM in PTM4Tag achieves the best performance among the five considered PTMs and outperforms the state-of-the-art Convolutional Neural Network-based approach by a large margin in terms of average Precision@k, Recall@k, and F1-score@k. We conduct an ablation study to quantify the contribution of a post’s constituent components (Title, Description, and Code Snippets) to the performance of PTM4Tag. Our results show that Title is the most important in predicting the most relevant tags, and utilizing all the components achieves the best performance.

CCS CONCEPTS
• Computing methodologies → Natural language processing; Information extraction.

KEYWORDS
Tag Recommendation, Transformer, Pre-Trained Models

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

© 2022 Copyright held by the owner/author(s). Publication rights licensed to ACM. ACM ISBN 978-1-4503-9298-3/22/05...$15.00 https://doi.org/10.1145/3524610.3527897

1 INTRODUCTION

Software question & answer (SQA) sites [29, 30, 32, 35, 37, 40] have emerged as essential resources for assisting software developers. Stack Overflow (SO), one of the largest SQA sites, features a powerful platform that facilitates collaboration and communications among developers in a wide range of programming-related activities. The site has accumulated extensive volumes of software engineering knowledge. As of January 2022, Stack Overflow has more than 17 million registered users and hosted over 22 million questions with 33 million answers.¹

The rapid growth of Stack Overflow highlights the need to efficiently manage the site’s content at a large scale and support queries from users. A standard solution employed by modern SQA sites is to allow users to tag their posts with one or a few technical terms, which play an essential role in structuring and navigating content. Thus, selecting accurate and appropriate tags that summarize the question post concisely can help many aspects of the site usage, e.g., connecting the expertise among different communities, delivering the question to the appropriate set of people with the right expertise, etc. In the early days, Stack Overflow set no restrictions on tagging posts; users were free to create and choose any tags with arbitrary input. However, the quality of tags highly depends

¹https://stackexchange.com/sites?view=list#traffic

Junda He, Bowen Xu, Zhou Yang, DongGyun Han, Chengran Yang, and David Lo. 2022. PTM4Tag: Sharpening Tag Recommendation of Stack Overflow Posts with Pre-trained Models. 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/10.1145/3524610.3527897
on users' level of expertise, English skills, writing styles and so on, making the tags selected by different users likely to be inconsistent, which triggers the tag synonym\(^2\) and tag explosion \([1]\) problems. Such problems emphasize the desirability of a tag recommendation technique that learns to automatically predict tags for an unseen post based on a large volume of historical posts.

The task of tagging posts can be characterized as a multi-label classification problem, where the goal is to select the most relevant subset of tags from a large group of tags. Tagging SO posts is considered challenging for several reasons. As the levels of expertise vary by person, allowing users to tag their posts introduces much noise and makes the tag set rather sparse. The posts in Stack Overflow cover an extensive range of topics (e.g., over 10 thousand available tags), making it challenging to build one model to accurately capture the semantics of the post and establish connections between the posts and the related tags.

There is a growing body of literature that focuses on the tag recommendation task of SO posts. Recently, Xu et al. \([37]\) have proposed Post2Vec, a deep learning-based approach that has been applied to the tag recommendation task and has been shown to achieve state-of-the-art performance over a number of deep-learning-based baselines \([40]\). Xu et al. adopted Convolutional Neural Networks (CNNs) \([22]\) as the feature extractors in Post2Vec. In this work, we focus on further improving the tag recommendation performance by leveraging the transformer-based pre-trained models (PTMs), which enhanced CNN with the self-attention mechanism and pre-trained knowledge.

The transformer-based PTMs such as BERT \([5]\) and RoBERTa \([16]\) have established great progress in the field of Natural Language Processing (NLP) and achieved phenomenal performance in several downstream tasks, i.e., Question Answering \([20]\), Text Classification \([10]\). Inspired by the success of PTMs in the NLP domain, there is increased interest in adapting pre-training models to the field of software engineering (SE) \([6, 26, 28]\). Such PTMs have been very effective in multi-class or pair-wise text classification tasks such as sentiment analysis \([39]\). API review \([38]\). However, not much SE literature has investigated the performance of PTMs in handling a multi-label classification problem with hundreds or thousands of labels. This motivated us to explore whether BERT-based PTMs could outperform the state-of-the-art SO tag recommendation approach \([37]\).

Furthermore, several studies have shown that domain-specific PTMs can perform significantly better than language models that are pre-trained on the general domain text in modeling the target domain \([2, 8, 13, 19]\). Generally speaking, text in different domains usually obeys a different word distribution, and general-purpose PTMs may fail to capture the correct semantics of technical jargon. To give an example in the SE domain, the word "Cookies" would refer to a small chunk of data stored in the user end by the web browser to support easy access to websites, and it does not refer to the "baked biscuit." However, a high-quality, large-scale corpus of a specialized domain usually is extremely difficult to obtain. General-purpose language models like BERT \([5]\) are usually trained on a vast amount of data which is usually significantly larger than domain-specific data. Considering that each kind of model has its potential strengths and weaknesses, it motivates us to explore the impact and limitations of adopting different PTMs.

In this work, we propose PTM4Tag, a framework that trains a BERT-based multi-label classifier to recommend tags for SO posts. To examine the effectiveness and contain a better understanding of PTM4Tag, we are interested in answering the following research questions:

**RQ1: Out of the five variants of PTM4Tag with different PTMs, which gives the best performance?** Considering that each model has its potential strengths and weaknesses, it motivates us to study the impact of adopting different PTMs in PTM4Tag. Hence, we investigate this research question by implementing PTM4Tag with different BERT-based models. Namely, we compare the results of domain-specific PTMs: CodeBERT \([6]\) and BERTOverflow \([26]\) with general domain PTMs: BERT \([5]\), RoBERTa \([16]\), and ALBERT \([12]\).

**RQ2: How is the performance of PTM4Tag compared to the state-of-the-art approach in Stack Overflow tag recommendation?** In this research question, we examine the effectiveness and performance of PTM4Tag by comparing it with the CNN-based state-of-the-art baseline for the tag recommendation task of Stack Overflow, i.e., Post2Vec \([37]\).**

**RQ3: Which component of post benefits PTM4Tag the most?** PTM4Tag is implemented with a triplet architecture, which encodes the three components of a SO post, i.e., Title, Description, and Code with different Transformer-based PTMs. Considering that each component may carry a different level of importance for the tag recommendation task, it inspires us to explore the contribution of each component by conducting an ablation study.

The contributions of the paper are as follows:

1. To the best of our knowledge, our work is the first to leverage BERT-based pre-trained language models for tag recommendation of SQA sites. We explore the effectiveness of different PTMs by training five variants of PTM4Tag and compared their performance by categorizing them into two groups, generic and domain-specific. Based on our findings, we advocate SE researchers considering CodeBERT as their first attempt or baseline approach for SQA-related tasks.

2. Our experiment results demonstrate that PTM4Tag with CodeBERT outperforms the state-of-the-art CNN-based approach by a large margin but PTM4Tag with ALBERT and BERTOverflow perform worse than the state-of-the-art approach. Thus, we conclude that leveraging PTM can help to achieve more promising results than the existing approach, but PTM within PTM4Tag needs to be rationally selected.

3. The results of our ablation study show that Title is the most significant component for tag recommendation of SO. Still, considering all components of a post would yield the best performance of PTM4Tag.

The paper is structured as follows: Section 2 introduces the background knowledge of the tag recommendation task of SO, the state-of-the-art baseline approach and five popular PTMs that are investigated in our study. Section 3 describes our proposed approach in detail, and then Section 4 specifies the experimental settings. Sections 5 presents the experimental results with analysis. In Section 6, we conducted a qualitative analysis, discussed the
threats to validity as well as summarized the lessons we learned from our experiment results. Section 7 reviews the literature on PTMs applied in SE, and the SQA-oriented tag recommendation approaches. Finally, in Section 8 we conclude our work and mention the future work.

2 BACKGROUND

In this section, we first formalize the task of recommending tags for SO post as a multi-label classification problem. Then, we describe Post2Vec [37], the current state-of-the-art tag recommendation approach. In the end, we briefly introduce the five pre-trained language models that are investigated in this paper.

2.1 Tag Recommendation Problem

Considering a SO post can be labeled by one or multiple tags, we regard the task of recommending tags for SO post as a multi-label classification problem. We assume to have a corpus of SO posts (denoted by $X$) and a collection of tags (denoted by $Y$). Formally speaking, given a SO post $x \in X$, the tag recommendation task aims to acquire a function $f$ that maps $x$ to a subset of tags $y = \{y_1, y_2, ..., y_L\} \subseteq Y$ that are most relevant to the post $x$. We denote the total number of training examples as $N$, the total number of available tags as $L$ and the number of tags of a training data as $l$, such that $L = |Y|$ and $l = |y|$. Noted that a SO post can be labeled with at most five tags, so $l$ must be no larger than 5.

2.2 The State-of-the-art Approach

Xu et al. proposed Post2Vec [37], a deep learning-based tag recommendation approach for SO posts, and achieved the current state-of-the-art performance. Xu et al. trained several variants of Post2Vec to examine the architecture design from multiple aspects. In this paper, we select their best-performing model as our baseline model. More specifically, it leveraged CNN as the feature extractors of the post and divided the content of a post into three components, i.e., Title, Description, and Code. Each component of a post has its own component-specific vocabulary and is modeled separately with a different neural network. A major drawback of Post2Vec is that their underlying neural assigns equal contribution to the input tokens. We have addressed this limitation by leveraging the Transformer-based PTMs, which enhanced the architecture with a self-attention mechanism and pre-trained knowledge obtained from other datasets. Additionally, the importance of each component remains unknown in the experiments of Post2Vec. We have further conducted an ablation study to investigate the contribution of each component to the task.

2.3 Pre-trained Language Models

Recent trends in the NLP domain have led to the rapid development of transfer learning. Especially, substantial work has shown that pre-trained language models learn practical and generic language representations which could achieve outstanding performance in various downstream tasks simply by fine-tuning, i.e., without training a new model from scratch [10, 15, 20]. With proper training manner, the model can effectively capture the semantics of individual words based on their surrounding context and reflect the meaning of the whole sentence.

In this section, we describe the five popular PTMs investigated in our experiments. We first introduce three large-scale, general-purpose PTMs. Namely, they are BERT [5], RoBERTa [16] and ALBERT [12]. Besides, we also consider two additional pre-trained language models designed for software engineering tasks (i.e., CodeBERT [6] and BERTOverflow [26]).

BERT. BERT [5] is based on the Transformer architecture [27] and contains the bidirectional attention mechanism. BERT is pre-trained on a large corpus of general text data, including the entire English Wikipedia dataset and the BooksCorpus [43]. It has two pre-training tasks: Masked Language Modeling (MLM) and Next Sentence Prediction (NSP). Given an input sentence where some tokens are masked out, the MLM task predicts the original tokens for those masked tokens. Given a pair of sentences, the NSP task aims to predict whether the second sentence in the pair is the subsequent sentence to the first sentence.

RoBERTa. RoBERTa [16] is mainly based on the original architecture of BERT but modifies a few key hyper-parameters. It removes the NSP task and feeds multiple consecutive sentences into the model. RoBERTa is trained with larger batch size and learning rate on a dataset that is an order of magnitude larger than the training data of BERT [5, 43].

ALBERT. ALBERT [12] is claimed as A Lite BERT. ALBERT involves two parameter reduction techniques: factorized embedding parameterization and cross-layer parameter sharing. Additionally, it replaces the NSP task used by BERT with the Sentence Order Prediction (SOP) task. By doing so, ALBERT can significantly reduce the number of model parameters and facilitate the training process without sacrificing the model performance.

CodeBERT. CodeBERT follows the same architectural design as RoBERTa. However, CodeBERT [6] is pre-trained on both natural language (NL) and programming language (PL) data from the CodeSearchNet database provided by [9]. CodeBERT considers two objectives at the pre-training stage: masked language modeling (MLM) and Replaced Token Detection (RTD). The goal of the RTD task is to identify which tokens are replaced from the given input. CodeBERT uses bimodal data (NL-PL pairs) as input at the pre-training stage. Thus, it understands both forms of data. The CodeBERT model has been proven practical in various SE-related downstream tasks, such as Natural Language Code Search [7, 42], program repair [17], etc [6].

BERTOverflow. BERTOverflow is trained by Tabassum et al. [26] as a domain-specific PTM on 152 million sentences from Stack Overflow. Notice that BERTOverflow The authors have introduced a software-related named entity recognizer (SoTfNER) that combines an attention mechanism with code snippets. The model follows the same design as the BERT model with 110 million parameters.

Recently, Mosel et al. [28] have proposed seBERT, which is pre-trained on textual data from Stack Overflow⁴, GitHub⁵, and Jira

---

⁴https://cloud.google.com/bigquery/
⁵https://www.garchive.org/
We transform the raw SO data into a desirable format at the pre-work for SO posts. This section introduces our proposed tag recommendation framework to be of low quality. The intuition is that users typically write on seBERT requires a much longer training time and more GPU processing stage. Each SO post would be decomposed into three respectively.

Besides, code snippets in different posts can cover a wide range of programming languages and frameworks, making it challenging to

Figure 1: An example of an SO Post. A post contains a short title that summarizes the main content of this post. The body of a post can include detailed descriptions written in natural languages and code snippets.

we have treated the Code as noise and discarded it during the pre-processing stage. However, we observe that the Code in an SO post provides valuable semantic information that can be leveraged to help in recommending tags more accurately. Take a SO post shown in Figure 1 as an example, one of its tags is ‘python’, but neither Title nor Description explicitly mentions the post is python-related. If we only look at the title and description sections, it is unclear which programming language this post is asking about. However, by only considering the Code, the grammar of Python can be easily used to infer that the post is likely to relate to the tag ‘python’. Motivated by this example, we further divide the Body into Description and Code, where the Description corresponds to the part of Body mainly written in natural languages and Code refers to the code snippets within the Body.

According to the web design of StackOverflow, the code snippets contained within the Body of a SO post are usually wrapped with HTML tags (<pre><code>...</code><\pre>). Thus, we first use a regular expression formula \"<pre><code>(\[\s\S\]*?)<\code><\pre>\" to split the Body of a SO post into sections of natural language and programming language. We then further remove the redundant HTML tags within these sections since these HTML tags are used for formatting and are irrelevant to the content of a post. By the above mean, we extract the Description and Code from each SO post. Following the above procedure, we consider that a post is made up of four constituents: the Title, Description, Code, and Tags. Our proposed PTM4Tag framework takes Title, Description and Code as input and aims to predict the sets of tags that are most relevant to this post (i.e., the Tags).

3 METHODOLOGY

This section introduces our proposed tag recommendation framework for SO posts. PTM4Tag in detail. The overall architecture of PTM4Tag with three stages is illustrated in Figure 2, which are Pre-processing, Feature Extraction, and Classification respectively. We transform the raw SO data into a desirable format at the pre-processing stage. Each SO post would be decomposed into three components: Title, Description, and Code snippets. Thus, PTM4Tag is implemented with a triplet architecture and leverages three PTMs as encoders to generate representations for each component. At the feature extraction stage, we feed the processed data into the used PTMs and represent each component as a feature vector. The obtained feature vectors are then fed to a fusion layer to construct the representation of the SO post. At the classification stage, the classifier maps the post representation to a tag vector that indicates the probability of each tag.

3.1 Pre-processing

3.1.1 Post Component Extraction. Figure 1 illustrates a typical SO post that consists of three components: Title, Body and Tags. The Title summarizes the question, and the Body provides more details and contexts to help other users understand the question. The Body of a SO post usually contains two parts: textual description and code snippets, which we refer to them as Description and Code in the following part of the paper.

Some researchers [14, 32, 40] consider the Code in an SO post to be of low quality. The intuition is that users typically write the post to seek help, which means that the code snippets within the post are likely to be written by novices and have low quality. Besides, code snippets in different posts can cover a wide range of programming languages and frameworks, making it challenging to

3.1.2 PTM-Oriented Tokenization. Since the design of PTM4Tag leverages BERT-based PTMs, we rely on the corresponding tokenizer of the underlying pre-trained model to generate token sequences. The BERT-based PTMs usually accept a maximum input sequence length of 512 sub-tokens which also always include two special tokens <CLS> and SEP. The <CLS> token (short for ClassSification) is the first token of every input sequence, and the corresponding hidden state is then used as the aggregate sequence representation. <SEP> token (short for SEParator) is inserted at the end of every input sequence. The problem of capturing long text arises since a significant proportion of the training point has exceeded the maximum acceptably input limit of the PTMs. According to our dataset statistics, more than 50% of the posts’ descriptions and more than 40% of the posts’ code snippets are longer than the given length limit. We tackle the problem using a head-only truncation strategy [24], which only considers the initial 510 tokens (excluding the <CLS> and <SEP> tokens) as the input tokens. There are other truncation strategies such as tail-only truncation, head+tail truncation [24] and other standard techniques to handle longer text such as compression [24], hierarchical method [24]. However, such approaches are not attempted in our work since there is no clear conclusion on which approach would give the best performance in general, and the performance varies among different situations. We leave the extensive exploration of the effect of such methods for future work.
3.2 PTM-based Feature Extraction

3.2.1 Language Modeling with PTMs. A language model learns the probability distribution of sequences of words. Recently, the BERT-based PTMs have been achieving outstanding results in a range of natural language understanding tasks [10, 20]. The BERT architecture [5, 27], which is also inherited in other models, e.g., RoBERTa and CodeBERT is essentially the Encoder stack of transformer [27]. It is implemented with the self-attention mechanism [27] which significantly enhances its ability in capturing long dependencies. The BERT_BASE model contains 12 layers and 110M parameters in total with 512 hidden units and 8 attention heads. BERT-based models usually are pre-trained on a large-scale corpus in order to obtain a generalized representation. Taking the expensive computational cost at the pre-training stage into account, we leveraged the released PTMs by the community in the design of PTM4Tag to generate contextual word embeddings. The encoder part of PTM4Tag is replaceable, and we have empirically implemented five variants of PTM4Tag with different PTMs (refer to Section 4.3) and investigate the impact of the PTM selection within PTM4Tag (refer to our first research question, i.e., RQ1 in Section 5).

3.2.2 Pooling and Fusion. After the word embeddings are generated, we obtain the post representation by applying a pooling strategy. A pooling strategy executes the down-sampling on the input representation and it is widely used to generate sentence embeddings from a sequence of word embeddings (wherein a BERT-based model, each word embedding has 768 dimensions). We generate post embeddings in the same way as generating sentence embeddings with BERT models. There are several common choices to derive fixed size sentence embeddings from a BERT-based model, include (1) using the first <CLS> token, (2) Average Pooling and (3) Maximum Pooling [21].

Reimers and Gurevych [21] have evaluated the effectiveness of different pooling strategies on the SNLI dataset [3] and the Multi-Genre NLI dataset [34] in the sentence classification task, and the reported Average Pooling gives the best performance in both datasets. Inspired by the findings, our proposed method leverages the Average Pooling strategy on the hidden output to generate component-wise feature vectors by default. Finally, we concatenate these three vectors sequentially to obtain the final representation of an SO post.

3.3 Model Training and Inference

After performing average pooling, the output of the fusion layer is fed into a feed-forward neural network to perform the task of multi-label classification. Given a training dataset consisting of $X$ and corresponding ground truth labels $y$ for each $x \in X$, we train a tag recommendation model $f$ by minimizing the following objective:

$$
L = -\frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{L} y_{ij} \log(f(y_{ij}|x_i)) + (1-y_{ij}) \log(1-f(y_{ij}|x_i))
$$

where $N = |X|$ is the total number of training examples and $f(y_{ij}|x_i)$ is the probability that tag $y_{ij}$ is related to SO post $x_i$. The objective captures the binary cross-entropy loss on all the training examples and can be optimized by gradient descent via back-propagation. Note that the gradient flow through the multi-label classifier as well as the PTMs used to process Title, Description, and Code. The parameters of both the tag predictor after the fusion layer and the PTMs are updated during the training process of PTM4Tag.

Given an input $x_i$, PTM4Tag produces a vector corresponding to all the tags. An element $f(y_{ij}|x_i)$ in the vector corresponds to the probability that tag $y_{ij}$ is relevant with SO post $x_i$. Stack Overflow sets a limit $k$ for the number of tags a post can have, and we rank the tags in descending order according to their probabilities produced by PTM4Tag. The top $k$ tags with the highest probabilities are recommended to a SO post.

4 EXPERIMENTAL SETTINGS

This section describes the dataset investigated in our experiment, the commonly-used evaluation metrics of a tag recommendation technique, and the implementation details of all considered models.

4.1 Data Preparation

To ensure a fair comparison to the current state-of-the-art approach [37], we select the same dataset as Xu et al. as the benchmark. The original data is retrieved from the snapshot of the Stack Overflow dump versioned on September 5, 2018. A tag is regarded as rare if its occurrence is less than a pre-defined threshold $\theta$. The intuition is that if a tag appears very infrequently in such a large corpus of posts (over 11 million posts in total), it is likely to be an incorrectly-created tag that developers do not broadly recognize. Therefore, we remove such rare tags as they may reduce the quality of the training and test data. Following the same procedure as [14, 37], we set the threshold $\theta$ for deciding a rare tag as 50, and

---

6https://archive.org/details/stackexchange
we remove all the rare tags of a post, and we remove the posts which contain rare tags only from the dataset. In the end, we have identified 29,357 rare tags and 23,687 common tags in total. As a result, we obtained a dataset consisting of 10,379,014 posts. We select 100,000 latest posts as the test data and used the rest of the 10,279,014 posts as the training data.

4.2 Evaluation Metrics

Previous studies of tag recommendation [14, 37, 40] on SQA sites use Precision@k, Recall@k, F1-score@k for evaluating the performance of the approaches. To be consistent, we follow the same evaluation metrics in this work. Formally speaking, given a corpus of SO posts, \( X = \{x_1, \ldots, x_n\} \), we report Precision@\( k_i \), Recall@\( k_i \), F1-score@\( k_i \) on each post \( x_i \) respectively where \( 0 \leq i \leq n \) and calculate the average of Precision@\( k_i \), Recall@\( k_i \), F1-score@\( k_i \) as Precision@\( k \), Recall@\( k \), F1-score@\( k \) to be the final measure.

Precision@\( k \) measures the average ratio of predicted ground truth tags among the list of the top-k recommended tags. For the \( i \)th post in the test dataset, we denote its ground truth tags of a particular post by \( GT_i \) and predicted top-k tags of the model by \( Taq^k_i \). We calculate Precision@\( k_i \) as:

\[
\text{Precision}@k_i = \frac{|GT_i \cap Taq^k_i|}{k}
\]

Then we average all the values of Precision@\( k_i \):

\[
\text{Precision}@k = \frac{\sum_{i=1}^{n} |GT_i \cap Taq^k_i|}{|X|}
\]

Recall@\( k \) reports the proportion of correctly predicted ground truth tags found in the list of ground truth tags. The original formula of Recall@\( k_i \) has a notable drawback: the Recall score would be capped to be small when the value of \( k \) is smaller than the number of ground truth tags. In the past literature [14, 37, 40], a modified version of Recall@\( k \) is commonly adopted as indicated in Equations 4 and 5. We have adopted the modified Recall@\( k \) in our work, which is as same as the one used to evaluate the current state-of-the-art approach in [57].

\[
\text{Recall}@k_i = \begin{cases} 
\frac{|GT_i \cap Taq^k_i|}{k} & \text{if } |GT_i| > k \\
\frac{|GT_i \cap Taq^k_i|}{|GT_i|} & \text{if } |GT_i| \leq k 
\end{cases}
\]

\[
\text{Recall}@k = \frac{\sum_{i=1}^{n} \text{Recall}@k_i}{|X|}
\]

F1-score@\( k \) is the harmonic mean of Precision@\( k \) and Recall@\( k \) and it is usually considered as a summary metric. It is formally defined as:

\[
\text{F1-score}@k_i = 2 \times \frac{\text{Precision}@k_i \times \text{Recall}@k_i}{\text{Precision}@k_i + \text{Recall}@k_i}
\]

\[
\text{F1-score}@k = \frac{\sum_{i=1}^{S} \text{F1-score}@k_i}{|X|}
\]

A large volume of literature on tag recommendation of SQA sites evaluates the result with \( k \) equals to 5, and 10 [14]. However, the number of the tags for Stack Overflow post is not allowed to be greater than 5; thus, we set the maximum value of \( k \) to 5, and we evaluate \( k \) on a set of values such that \( k \in \{1, 2, 3, 4, 5\} \).

4.3 Implementation

To answer the three research questions mentioned in Section 1, we trained eight variants of PTM4Tag. Details about each variant model is summarized in Table 1.

For RQ1, we trained five variants of PTM4Tag by using different PTMs as the encoders (i.e., CodeBERT\(_{ALL} \), ALBERT\(_{ALL} \), BERT\(_{ALL} \), RoBERTa\(_{ALL} \), BERTOverFlow\(_{ALL} \) in Table 1) and empirically investigate their performance. More specifically, each variant follows the triplet architecture as illustrated in Figure 2.

For RQ2, we compared the performance of PTM4Tag (with the best performing PTM) with the state-of-the-art approach, namely Post2Vec. To reproduce the baseline introduced in Section 2.2, we re-use the replication package \(^7\) released by the original authors.

\(^7\)https://github.com/maxxbw54/Post2Vec

\(^8\)https://huggingface.co

\(^9\)https://huggingface.co
5 EXPERIMENTAL RESULTS

In this section, we conduct experiments to evaluate the performance of the five variants of our proposed framework and the baseline approach. We further conduct an ablation study on our best-performing variant. We present the mean values of Precision@k, Recall@k, and F1-Score@k for each model. Based on the results, we answer the research questions presented in Section 1.

RQ1. Out of the five variants of PTM4Tag with different PTMs, which gives the best performance?

Motivation BERT-based pre-trained language models have witnessed great success across multiple SE-related tasks. To the best of our knowledge, our work is the first that leverages PTMs in recommending tags for SOQ sites. Past studies have shown that different PTMs have different strengths and weaknesses. For example, Mosel et al. [28] found BERToverflow outperforms the general-purpose PTM, BERT, by a substantial margin in issue type prediction and the commit intent prediction. Mosel et al. further reported that the vocabulary of SE domain-specific PTMs (i.e., BERToverflow) contain many programming-related vocabularies such as jvm, bugzilla and debug which are absent in the vocabulary of BERT [5]. However, Yang et al. reported that the generic PTMs BERT performs better than domain-specific models, i.e., BERToverflow in API review classification task [38]. Domain-specific PTMs may understand the text from the target domain better, but general-purpose data is likely to be pre-trained on larger data. Thus, the efficacy of pre-trained models on this task remain unclear. Since the underlying BERT-based PTM of PTM4Tag is replaceable, it evokes our interest in investigating the effectiveness of different PTMs under the PTM4Tag architecture and finding the most suitable PTM for SO posts tag recommendation.

Results and Analysis To answer the question, we report the performance of five variants of PTM4Tag, which are implemented with different PTMs, which are three general-purpose PTMs (BERT, RoBERTa, and ALBERT) and two SE-specific PTMs (CodeBERT and BERToverflow). These variants are implemented with the Triplet architecture, which considers Title, Description, and Code as input. We refer to these variant models as BERTALL, RoBERTALL, ALBERTALL, CodeBERTALL, and BERTOverfowALL, respectively.

Table 2 illustrates the obtained results of all variants. We observed that leveraging CodeBERT consistently outperformed other models in all evaluation metrics. BERTALL and RoBERTALL showed worse performance than CodeBERTALL only by a small margin. In terms of F1-score@5, CodeBERTALL, BERTALL, and RoBERTALL achieved 0.513, 0.510 and 0.505, respectively. BERTOverfowALL performed slightly better than ALBERTALL, and ALBERT is the worst-performing model. BERToverfowALL and ALBERTALL only achieved 0.427 and 0.406 in F1-score@5, which are lower than CodeBERT, BERT and RoBERTa by a large margin.

The potential reason for why CodeBERTALL achieved the best performance is that it is a bi-modal PTM for SE domain that has leveraged both natural language and programming language at the pre-training stage. Differently, the other models are trained with natural language data only. Since PTM4Tag uses both natural and programming languages as input for tag recommendations, the nature of the input of the task well matches with CodeBERT.

ALBERTALL and BERToverfowALL are largely outperformed by the rest of the variants. A potential reason could be that ALBERT includes fewer parameters since the design of ALBERT aspires to address the GPU memory limitation. BERToverfow follows the same architecture as BERT, but BERTALL performs better than BERToverfowALL by a large margin. By intuition, BERToverfow is formulated with a much more appropriate vocabulary for the SE domain, which would also produce promising results, but the experimental results may have indicated that BERToverfow still required more training. It is potentially caused by the quality and the size of the datasets used at the pre-training stage. BERToverfow is pre-trained with 152 million sentences from SO. BERT is trained on the entire English Wikipedia and the Book Corpus dataset, written by professionals and constantly reviewed. However, sentences from SO could be written by arbitrary authors and the existence of in-line code within a post would introduce extra noise. Additionally, the training corpus of BERT contains 3.3 billion words in total, and the average sentence length of BookCorpus is 11 words. By estimation, the training corpus of BERT is likely to be twice more than BERToverfow.

Although the CodeBERT and BERToverfow both are SE domain-specific PTMs, fundamentally, are different from each other. In architecture-wise, CodeBERT is initialized with RoBERTa’s parameters and adds a second phase of pre-training on the CodeSearchNet dataset [9], whereas BERToverfow is trained from scratch. Moreover, both BERT and RoBERTa are models designed for natural language only. Although we include code snippets as input, BERT
and RoBERTa still achieve outstanding results (slightly worse than CodeBERT). It possibly indicates that pre-trained models trained with a large scale of natural language data also be helpful for programming language modeling.

**Answers to RQ1:** Among the five considered PTMs of PTM4Tag, the one implemented with CodeBERT produces the best performance.

### RQ2. How is the performance of PTM4Tag compared to the state-of-the-art approach in Stack Overflow tag recommendation?

**Motivation** The current state-of-the-art approach for recommending tags of SO posts is implemented based on Convolutional Neural Network and trained from scratch [37]. However, Transformer-based PTMs are strengthened with the self-attention mechanism and transferred pre-train knowledge. In this research question, we investigate whether the variants of PTM4Tag can achieve better performance than the current state-of-the-art approach.

**Results and Analysis** As presented in Table 2, the best performing variant of PTM4Tag, i.e., CodeBERT, substantially outperformed Post2Vec in terms of F1-score@k (k from 1 to 5) by 7.9%, 11.3%, 13.8%, 15.0% and 15.3%, respectively. Furthermore, BERT and RoBERTa also surpassed Post2Vec. BERT improved the F1-score@1-5 of Post2Vec by 7.5%, 10.5%, 13.1%, 14.1% and 14.6%; and RoBERTa by 7.3%, 10.1%, 12.4%, 13.3% and 13.5%; However, we also observe that not all PTMs are able to achieve outstanding performance under PTM4Tag. Post2Vec outperformed BERT/Overflow by 8.4%, 6.6%, 8.5%, 2.7%, 4.7% and outperformed ALBERT by 5.1%, 7.7%, 8.5%, 9.2%, 9.6% in F1-score@1-5.

Different from Post2Vec, PTM4Tag involves a vast amount of knowledge accumulated from the dataset used for pre-training. PTM4Tag leverages PTMs to extract feature vectors and optimize the post representation during the fine-tuning stage, whereas Post2Vec learns post representations from scratch. Thus, PTMs provide a more decent initial setting. Furthermore, CodeBERT provides in-domain knowledge of SE. Our results indicate that the knowledge accumulated from the dataset used for pre-training.

**Motivation** PTM4Tag is designed with a triplet architecture where each component of a post, i.e., Title, Description, and Code, are modeled by utilizing separate PTMs. Title, Description, and Code snippets complement each other and describe the post from its own perspective. Title summarizes the question with a few words; Description further expands the content from the Title; Code snippets often is a real example of the problem. Thus it motivates us to investigate which component produces the most critical contribution in the PTM4Tag framework.

**Results and Analysis** To answer this research question, we conduct an ablation study to investigate the importance of each component, i.e., Title, Description, and Code, respectively. Note that PTM4Tag is implemented with a triplet architecture by default. To answer this research question, we modified it to a twin architecture to fit two considered components at a time. We train three ablated models with our identified best-performing PTM, i.e., CodeBERT.

The results for RQ3 are presented in Table 3. From the table, we identified that CodeBERT remained the best performing model on all the evaluation metrics. To provide a more intuitive understanding of the result, we further illustrate the performance gap of F1-score@1-5 between the ablated models and CodeBERT by visualizing in Figure 3, where the value on the y axis is calculated using the score of CodeBERT minus the score of the ablated model. CodeBERT yielded the most promising performance among the ablated models, which implies that the code snippets are beneficial, but they are the least significant among all three components.

An interesting finding is that CodeBERTNoDesc performed better in F1-score@1-3 and CodeBERTNoTitle performed better in F1-score@4-5, which suggests that Title is more important for boosting the performance of F1-score@1-3 and Description is more critical for improving F1-score@4-5. A possible explanation could
We conduct qualitative analysis to illustrate the capability of our
where
which could directly help the system predict the top few tags. De-
for our task.
that the pre-trained knowledge learned by CodeBERT is beneficial
Moreover, we found that the CodeSearchNet database [9] which is
occurred several times in the post, Post2Vec still failed to capture its
as to the ground truth tags. Although the word memoization has occurred several times in the post, Post2Vec still failed to capture its existence and the connection between memoization and decorator. Moreover, we found that the CodeSearchNet database [9] which is used to pre-train CodeBERT includes source code files that relate to both memoization and decorator. 11 This potential could indicate that the pre-trained knowledge learned by CodeBERT is beneficial for our task.

Figure 3: A line chart demonstrates performance difference in F1-score@k between each ablated models and CodeBERT_ALL, where k ∈ {1, 2, 3, 4, 5}. The value on the y axis is calculated using the corresponding score of the candidate ablated model minus the corresponding score of CodeBERT_ALL.

6 DISCUSSION
6.1 Error Analysis

We conduct qualitative analysis to illustrate the capability of our proposed framework PTM4Tag. Take a Stack Overflow post10 titled Pass Input Function to Multiple Class Decorators in the test dataset as an example. The ground truth tags of the post are decorator, memoization, python, python-2.7, and python-decorators. The tags predicted by Post2Vec are class, timer, python-decorators, decorator, and python while PTM4Tag gives the exact prediction as to the ground truth tags. Although the word memoization has occurred several times in the post, Post2Vec still failed to capture its existence and the connection between memoization and decorator. Moreover, we found that the CodeSearchNet database [9] which is used to pre-train CodeBERT includes source code files that relate to both memoization and decorator. 11 This potential could indicate that the pre-trained knowledge learned by CodeBERT is beneficial for our task.

��https://github.com/nerdvegas/rez/blob/1d3b846d53b5b5404edfe8ddb9083f9ceec8c5e7/src/rez/utils/memcached.py#L248-L375

6.2 Threats to Validity

Threats to internal validity. To ensure we implement the baseline (i.e., Post2Vec) correctly, we reused the official replication package released by the Xu et al.12 To instantiate variants of PTM4Tag with different pre-trained models, we utilized a widely-used deep learning library Hugging Face.13 Similar to prior studies [31, 33, 37], our work assumes that the tags are labeled correctly by users in Stack Overflow. However, some tags are potentially mislabelled. Still, we believe that Stack Overflow’s effective crowdsourcing process helps to reduce the number of such cases, and we further minimize this threat by discarding rare tags and posts (as described in Section 4.1).

Threats to external validity. We analyzed Stack Overflow, the largest SQA site, with a massive amount of questions. These questions cover diverse discussions on various software topics. As software technologies evolve fast, our results may not generalize to those newly emerging topics. Instead, our framework can adapt to new posts by fine-tuning models on more and new questions.

Threats to construct validity. Threats to construct validity are related to the suitability of our evaluation metrics. Precision@k, Recall@k, and F1-score@k are widely used to evaluate many tag recommendation approaches in software engineering [31, 33, 36, 37]. Thus, we believe the threat is minimal.

6.3 Lessons Learned

Lesson #1. Pre-trained language models are effective in tagging SO posts. The tag recommendation task for SQA sites has been extensively studied in the last decade [14, 32, 37, 40]. Researchers have tackled the problem via a range of techniques, e.g., collaborative filtering [14] and deep learning [37]. Furthermore, these techniques usually involve separate training of each component. Our experimental results have demonstrated that simply fine-tuning the pre-trained Transformer-based model can achieve state-of-the-art performance, even if there are thousands of tags. CodeBERT, BERT, and RoBERTa are capable of providing promising results for tag recommendation. Even though BERT and RoBERTa did not leverage programming language at the pre-training stage. Although PTMs are already widely adopted in SE tasks, most tasks are formulated as either binary classification problems or multi-class classification problems. In binary or multi-class classification problems the label classes are mutually exclusive, whereas for multi-label classification problems each data point may belong to several labels simultaneously. We encourage practitioners to leverage pre-trained models in the multi-label classification settings where the size of the label set could go beyond thousands. Moreover, our experiments also validate the generalizability of pre-trained models. We recommend practitioners apply pre-trained models in more SE tasks and consider fine-tuning pre-trained models as one of their baselines.

Lesson #2. CodeBERT is recommended as one of the first attempts for SQA-related tasks which involve both natural language and programming language. CodeBERT is trained on both

10https://stackoverflow.com/questions/51910978
11https://github.com/maxxbw54/Post2Vec
12https://github.com/nerdvegas/rez/blob/1d3b846d53b5b5404edfe8ddb9083f9ceec8c5e7/
src/rez/utils/memcached.py#L248-L375
13https://huggingface.co/

Answers to RQ3: Under PTM4Tag, Title and Description are more important than Code for tag recommendation. Title plays the most important role in predicting the top 3 most relevant tags, and the contribution of Description increases when the number of predicted tags increases from 4. Still, considering all the components can achieve the best performance.
bimodal and unimodal data. Our results showed that CodeBERT consistently outperforms other pre-trained models as well as the recent state-of-the-art deep learning model in tag recommendation task of Stack Overflow. We found that SE domain-specific model, i.e., CodeBERT is powerful, and researchers are recommended to use CodeBERT as one of their first attempts when their tasks are based on SQA data and involve both NL and PL.

Lesson #3. All components of a post from Stack Overflow are valuable pieces of semantics. Most previous literature has removed the code snippets from the pre-training process because they are considered noisy, poorly structured, and written in many different programming languages [14, 32, 40, 41]. However, our results show that code snippets are also beneficial to capture the semantics of SO posts and further boost the performance of the tag recommendation task. We encourage researchers to consider both the natural and programming languages parts of a post when analyzing SQA sites.

7 RELATED WORK

7.1 Pre-trained Models in SE

Recent works [2, 8, 13] have shown that the in-domain knowledge acquired by PTMs is valuable in improving the performance on domain-specific tasks, such as ClinicalBERT [8] for clinical text, SciBERT [2] for scientific text, and BioBERT [13] for biomedical text. Evoked by the success of PTMs in other domains, researchers start the work in the SE domain-specific PTMs [4, 6, 11, 26, 28]. Developers are free to create and pick arbitrary identifiers, making texts in the SE field frequently involve technical jargon and tokens from the programming languages [23]. This makes models pre-trained on general texts (e.g., BERT [5]) unsuitable to represent texts in the software engineering domain. Tabassum et al. [26] provided BERTOverflow for completing tasks related to Stack Overflow. BERTOverflow [26] inherits the architecture from BERT [5] and is trained on more than 152 million programming-related questions extracted from Stack Overflow. Mosel et al. [28] aims to provide a better SE domain-specific pre-trained model than BERTOverflow [26] and propose seBERT [28]. Since seBERT is developed upon the BERTLARGE architecture, we did not investigate the effectiveness of seBERT under the PTM4Tag framework due to restraints on computational resources and GPU memory consumption.

The aforementioned PTMs focus on modeling the natural language texts in the SE domain. In addition, there is a proliferation of studies on model programming languages with PTMs. Svyatkovskiy et al. trained GPT-C [25], a multi-layer generative transformer model, and leveraged GPT-C to build a code completion tool. GPT-C is pre-trained on 1.2 billion lines of source code written by multiple programming languages, including Python, JavaScript, C-sharp, etc. Feng et al. proposed CodeBERT [6], a bi-model pre-trained model for both natural and programming languages (NL and PL). CodeBERT is trained on bi-modal data of NL-PL pairs and a vast amount of uni-modal data. CodeBERT has two pre-training tasks: Masked Language Modeling (MLM) and Replacement Token Detection (RTD). The experimental results show that CodeBERT achieves the state-of-the-art performance on natural language code search and code document generation, and in a zero-shot setting, CodeBERT persistently outperforms RoBERTa [16].

7.2 Tag Recommendation for SQA Sites

Researchers have already extensively studied the tag recommendation task in the SE domain and proposed a number of approaches. Wang et al. introduced EnTagRec [29] that utilizes Bayesian inference and an enhanced frequentist inference technique. Results show that it outperformed TagCombine by a significant margin. Wang et al. then further extends EnTagRec [29] to EnTagRec++ [30], the latter of which additionally considers user information and an initial set of tags provided by a user. Zhou et al. proposed TagMulRec [41], a collaborative filtering method that suggests new tags of a post based on the results of semantically similar posts. Li et al. proposed TagDC [14], which is implemented with two parts: TagDC-DL that leverages a content-based approach to learn a multi-label classifier with a CNN Capsule network, and TagDC-CF that utilizes collaborative filtering to focus on the tags of similar historical posts. Post2Vec [37] distributively represents SO posts and is shown useful in tackling numerous Stack Overflow-related downstream tasks. We select Post2Vec [37] as the baseline in our experiments as it achieves the state-of-the-art performance in the tag recommendation task for SO posts.

8 CONCLUSION AND FUTURE WORK

In this work, we modeled the tag recommendation task of SO as a multi-label classification problem and introduced PTM4Tag, a pre-trained model-based framework for tag recommendation. We implemented five variants of PTM4Tag with different PTMs as the underlying Encoder. Our experiment results showed that the best variant of PTM4Tag is CodeBERT-based, and it outperforms the state-of-the-art approach by a large margin in terms of F1-score@5. Leveraging BERT and RoBERTa could also achieve desirable performance that only slightly loses CodeBERT and beat the state-of-the-art approach by a large margin. However, PTM4Tag models implemented with BERTOverflow and ALBERT did not give promising results. Even though the BERT-based PTMs are shown to be powerful and effective, PTMs behave differently, and the selection of PTMs needs to be carefully decided. Besides, we conducted an ablation study to investigate the contribution of each component under PTM4Tag. The result shows that Title is the most vital part for recommending the most relevant tags, and considering all components simultaneously and separately can produce the best performance. In the future, we plan to consider more diverse information of a post into account, such as the attached pictures, author information, etc. Also, we are interested in applying PTM4Tag on more SQA sites such as AskUbuntu14, etc., to further evaluate its effectiveness and generalizability. We release our replication package15 to facilitate future research.

ACKNOWLEDGMENTS

This research / project is supported by the National Research Foundation, Singapore, under its Industry Alignment Fund – Pre-positioning (IAF-PP) Funding Initiative. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not reflect the views of National Research Foundation, Singapore.

14https://askubuntu.com/
15https://github.com/soarsmu/PTM4Tag
