Empirical Software Engineering (2025) 30:28
https://doi.org/10.1007/s10664-024-10576-z

PTM4Tag+: Tag recommendation of stack overflow posts with pre-trained models

Junda He[1] · Bowen Xu[2] · Zhou Yang[1] · DongGyun Han[3] · Chengran Yang[1] · Jiakun Liu[1] · Zhipeng Zhao[4] · David Lo[1]

Accepted: 10 October 2024 / Published online: 20 November 2024

Abstract
Stack Overflow is one of the most influential Software Question & Answer (SQA) websites, hosting millions of programming-related questions and answers. Tags play a critical role in efficiently organizing the contents on Stack Overflow and are vital to support various site operations, such as querying relevant content. Poorly chosen tags often lead to issues such as tag ambiguity and tag explosion. Therefore, a precise and accurate automated tag recommendation technique is needed. Inspired by the recent success of pre-trained models (PTMs) in natural language processing (NLP), we present *PTM4Tag+*, a *tag recommendation framework* for Stack Overflow posts that utilize PTMs in language modeling. *PTM4Tag+* is implemented with a triplet architecture, which considers three key components of a post, i.e., Title, Description, and Code, with independent PTMs. We utilize a number of popular pre-trained models, including BERT-based models (e.g., BERT, RoBERTa, CodeBERT, BERTOverflow, and ALBERT), and encoder-decoder models (e.g., PLBART, CoTexT, and CodeT5). Our results show that leveraging CodeT5 under the *PTM4Tag+* framework achieves the best performance among the eight considered PTMs and outperforms the state-of-the-art Convolutional Neural Network-based approach by a substantial margin in terms of average $Precision@k$, $Recall@k$, and $F1\text{-}score@k$ (k ranges from 1 to 5). Specifically, CodeT5 improves the performance of $F1\text{-}score@1\text{-}5$ by 8.8%, 12.4%, 15.3%, 16.4%, and 16.6%, respectively. Moreover, to address the concern with inference latency, we experimented *PTM4Tag+* using smaller PTM models (i.e., DistilBERT, DistilRoBERTa, CodeBERT-small, and CodeT5-small). We find that although smaller PTMs cannot outperform larger PTMs, they still maintain over 93.96% of the performance on average while reducing the mean inference time by more than 47.2%.

Keywords Tag recommendation · Stack overflow · Pre-trained models · Transformer

1 Introduction

Stack Overflow (SO) is the largest online Software Question & Answer (SQA) platform that facilitates collaboration and communication among developers for a wide range of

Communicated by: Venera Arnaoudova,Sonia Haiduc,Gabriele Bavota

This article belongs to the Topical Collection: *International Conference on Program Comprehension (ICPC)*.

Extended author information available on the last page of the article

programming-related activities. As of January 2023, Stack Overflow has more than 20 million registered users and hosts over 23 million questions with 34 million answers.[1] Stack Overflow has accumulated extensive resources to assist software developers in their daily development processes.

The rapid growth of Stack Overflow highlights the need to manage the site's contents at a large scale. To address this challenge, Stack Overflow uses *tags* to categorize and structure the questions. Tags describe the topics and provide a concise summary of the question. Selecting accurate and appropriate tags can benefit various aspects of site usage, e.g., connecting the expertise among different communities, triaging questions to the appropriate people with relevant expertise, and assisting users in searching related questions (Maity et al. 2019; Zhou et al. 2019; Wang et al. 2015). However, tags on the site are constructed through the process known as folksonomy,[2] where the majority of tags are generated by users. The quality of tags largely depends on users' expertise levels, English proficiency, and other factors. Tags selected by different users are likely to be inconsistent, triggering problems like tag ambiguity (i.e., the same tag is used for various topics) (Maity et al. 2019) and tag explosion (i.e., multiple tags are used for the same topic) (Barua et al. 2014). As a result, users may struggle to find relevant questions, reducing the productivity of programmers. These negative impacts motivate researchers to develop an automated tag recommendation technique to recommend high-quality tags for questions.

In this paper, we characterize the task of tagging SO posts as a multi-label classification problem following previous literature (Wang et al. 2014; Zhou et al. 2019; Xu et al. 2021), i.e., selecting the most relevant subset of tags from a large group of tags. Tagging SO posts is considered challenging for several reasons. First, posts on Stack Overflow cover an extensive range of topics, resulting in an extremely large set of tags (over 10 thousand available tags). Second, allowing users to freely tag their posts introduces great inconsistencies and makes the tag set rather sparse. Building a single model to accurately capture post semantics and establish connections to relevant tags is challenging.

A growing body of literature tackled the *tag recommendation task* of SO posts (Zhou et al. 2019; Xu et al. 2021). The state-of-the-art solution for the task is Post2Vec (Xu et al. 2021). Post2Vec is a deep learning-based tag recommendation approach using Convolutional Neural Networks (CNNs) (Schmidhuber 2015) as the feature extractors. Motivated by the success of CNN in the SO post-tag recommendation task, we focus on further improving the tag recommendation performance by leveraging the transformer-based pre-trained models (PTMs). Compared to Post2Vec, PTMs enhance CNN with the self-attention mechanism and provide a much better model initialization with pre-training knowledge. Such properties make PTMs suitable for tagging SO posts and prompt our interest in adopting them.

Transformer-based PTMs have achieved phenomenal performance in the Natural Language Processing (NLP) domain (Devlin et al. 2018; Liu et al. 2019b; Raffel et al. 2020). They are shown to vastly outperform other techniques like LSTM (Huang et al. 2015) and CNN (Schmidhuber 2015). Inspired by their success, there is an increased interest in applying PTM to the field of software engineering (SE). Such PTMs are proven to be effective in multi-class or pair-wise text classification tasks such as sentiment analysis (Zhang et al. 2020) and API review (Yang et al. 2022). To the best of our knowledge, aside from our conference paper that this paper is extended from, no studies in the SE literature have investigated the performance of directly fine-tuning PTMs in handling a multi-label classification problem

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

[2] https://en.wikipedia.org/wiki/Folksonomy

with thousands of labels. This motivates us to explore the effectiveness of PTMs in the task of SO tag recommendation.

Nonetheless, directly applying PTMs from the NLP-domain to SE-related downstream tasks has limitations. Texts in different domains (i.e., NLP and SE) usually have different word distributions and vocabularies. Software developers are free to create identifiers they prefer when writing code. The formulated identifiers are often compound words and arbitrarily complex, e.g., *addItemsToList* (Shi et al. 2022). Words may also have a different meaning for software engineering. For example, the word "Cookie" usually refers to a small chunk of data stored by the web browser to support easy access to websites for software engineers, and it does not refer to the "baked biscuit" in the context of SE. PTMs trained with natural language text may fail to capture the semantics of SE terminology. NLP-domain language models (e.g., BERT Devlin et al. 2018 and RoBERTa Liu et al. 2019b) are usually extensively pre-trained with a significantly larger corpus than the SE-domain PTMs as it is much more difficult to obtain a high-quality, large-scale corpus of a specialized domain. Since each kind of model has its potential strengths and weaknesses, it prompts our interest in exploring the impact and limitations of different PTMs from the NLP-domain and SE-domain.

In this paper, we introduce *PTM4Tag+*, a framework that utilizes popular pre-trained models and trains a multi-label classifier on the transformer architecture for recommending tags for SO posts. We evaluate the performance of *PTM4Tag+* with different PTMs to identify the best variant. We categorize PTMs from two dimensions: domain-difference (NLP-domain vs. SE-domain) and architecture-difference (encoder-only vs. encoder-decoder). While the encoder-only BERT-based PTMs have been widely used in language modeling, we delve into the language representation from another family of pre-trained models: the encoder-decoder models. Encoder-decoder models like CodeT5 (Wang et al. 2021) and PLBART (Ahmad et al. 2021) boost the state-of-art-performance for numerous SE-related downstream tasks (e.g., code summarization, code generation, and code translation). Intuitively, encoder-decoder models should also be powerful in language modeling. Recent studies have demonstrated that the embeddings generated by T5 are superior to those generated by BERT (Ni et al. 2022). Nonetheless, in the SE domain, there are limited studies on leveraging popular encoder-decoder models (e.g., CodeT5 and PLBART) in classification tasks and evaluating the representation generated by these models. To fill this gap, we examine the efficacy of popular encoder-decoder models under the *PTM4Tag+* framework.

Furthermore, in order to increase the usability of *PTM4Tag+*, we reduce the size of *PTM4Tag+* by adopting smaller PTMs. In terms of deployment, a tool with a smaller size is more practical for integration in modern applications as it requires less storage and runtime memory consumption. On the other hand, the inference latency would be much faster, and user satisfaction can be hugely increased in the context of tagging SO posts. To address the concern with model size, we first identify several PTMs that can yield promising results under the *PTM4Tag+* framework and then leverage the smaller version of these PTMs to train four more variants of *PTM4Tag+* .

To obtain a comprehensive understanding of *PTM4Tag+*, we answer the following research questions:

RQ1: Out of the eight 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+*. Namely, we compare the results of BERT (Devlin et al. 2018), RoBERTa (Liu et al. 2019b), ALBERT (Lan et al. 2020b), CodeBERT (Feng et al. 2020), BERTOverflow (Tabassum et al. 2020), PLBART (Ahmad et al. 2021), CoTexT (Phan et al. 2021), and CodeT5 (Wang et al. 2021).

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 (Xu et al. 2021).

RQ3: Which component of post benefits PTM4Tag+ the most? *PTM4Tag+* is implemented with a triplet architecture, which encodes the three components of an 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, we explore the contribution of each component by conducting an ablation study.

RQ4: How is the performance of PTM4Tag+ with smaller pre-trained models? Utilizing PTMs gives great performance but results in high inference latency and difficulties in deployment (Shi et al. 2023). In this RQ, we select smaller PTMs to train *PTM4Tag+*.

As an extended version of our previous work (He et al. 2022), which proposed and evaluated the *PTM4Tag* framework, this paper further enhances *PTM4Tag* by empirically experimenting with more PTMs, including a set of Seq2Seq models and smaller PTMs.

We derived several useful findings from the experiments and we highlight the difference to our previous work in **bold**:

1. ***PTM4Tag+* with CodeT5 outperforms other PTMs and the previous state-of-the-art approach (i.e. Post2Vec) by a large margin.**
2. Although *PTM4Tag+* with ALBERT and BERTOverflow perform worse than Post2Vec, other PTMs are capable of outperforming Post2Vec. Thus, we conclude that leveraging PTMs can help to achieve promising results, but the PTM within *PTM4Tag+* needs to be rationally selected.
3. **Encoder-decoder PTMs can also benefit the *PTM4Tag+* framework. While encoder-decoder PTMs are usually used for generation tasks, we advocate future studies to include the Seq2Seq PTMs as baseline methods for representing SO posts.**
4. **We reduce the size of *PTM4Tag+*. Specifically, the inference time is improved by more than 47.2%, while the model preserves more than 93.96% of the original performance.**

The contributions of the paper are summarized as follows:

1. We propose *PTM4Tag+*, a transformer-based muli-label classifier, to recommend tags for SO posts. To the best of our knowledge, our work is the first to leverage pre-trained language models for tag recommendation of SO posts. Our proposed tool outperforms the previous state-of-the-art technique, Post2Vec, by a substantial margin.
2. We explore the effectiveness of different PTMs by training eight variants of *PTM4Tag+* and comparing their performance. We consider a set of BERT-based PTMs and **encoder-decoder PTMs**.
3. We further conducted an ablation study to investigate the contribution of each component to the task.
4. **We address the concern with model size and inference latency by involving a set of smaller PTMs under the *PTM4Tag+* framework.**

The paper is structured as follows: Section 2 introduces the background knowledge for 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. Section 4 specifies the experimental settings. Section 5 presents the experimental results with analysis. In Section 6, we conducted a qualitative analysis, discussed the threats to validity

and summarized our learned lessons. Section 7 reviews the literature on PTMs applied in SE, and the tag recommendation approaches for Software Question & Answer sites. Finally, we conclude our work and discuss future work in Section 8.

2 Background

In this section, we formalize the tag recommendation task for SO posts as a *multi-label classification problem*. Then, we describe Post2Vec (Xu et al. 2021), the state-of-the-art tag recommendation approach that is used as a baseline in this paper. In the end, we introduce the pre-trained language models that are leveraged in *PTM4Tag+*.

2.1 Tag Recommendation Problem

Considering that an SO post can be labeled by one or multiple tags, we regard the tag recommendation task for SO posts as a *multi-label classification problem*. We denote the corpus of SO posts by $\mathcal{X}$ and the collection of tags as $\mathcal{Y}$. Formally speaking, given an SO post $x \in \mathcal{X}$, the tag recommendation task aims to acquire a function f that maps x to a subset of tags $y = \{y_1, y_2, ..., y_l\} \subset \mathcal{Y}$ that are most relevant to the post x. We denote the total number of training examples as N, the total number of available tags[3] as L and the number of tags of training data as l, such that $L = |\mathcal{Y}|$ and $l = |y|$. Note that one SO post can be labeled with at most five tags, so l must be equal to or less than 5.[4]

2.2 Post2Vec

Xu et al. proposed Post2Vec (Xu et al. 2021), a deep learning-based tag recommendation approach for SO posts, which achieves 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 variant as our baseline model. Specifically, our baseline model leverages CNN as the feature extractor of the post and divides 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.

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 them on a smaller dataset, i.e., without training a new model from scratch (Lin et al. 2021; Jin et al. 2020; Qu et al. 2019). 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.

A major drawback of Post2Vec is that its underlying neural network (i.e., CNN) has limitations in modeling long input sequences. CNN requires large receptive fields to model long-range dependencies (Schmidhuber 2015). However, increasing the receptive field dra-

[3] https://stackoverflow.com/help/tagging

[4] https://resources.stackoverflow.co/topic/product-guides/topic-tag-targeting/

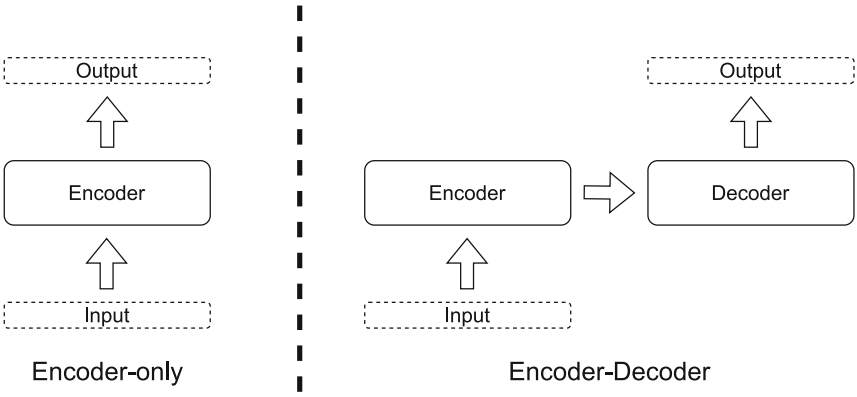

Fig. 1 The architecture of encoder-only and encoder-decoder models

matically reduces computational efficiency. We 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. We categorized the considered PTMs in this paper into two types: the encoder-only PTMs and the encoder-decoder PTMs. The architectural difference between these two types of PTMs is illustrated in Fig 1. In Table 1, we summarize the architecture, pre-training tasks, downstream tasks from the original paper, and language type of the PTMs used in this paper. Tables 2 and 3 presents the details of the abbreviation used in Table 1.

2.3.1 BERT-based Pre-trained Models

BERT-based Pre-trained Models utilize only the encoder stacks of a Transformer model. The attention layers have the ability to access every word of the input sentence, and these models are commonly referred to as having “bi-directional" attention. The pre-training objectives of these models typically involve some form of corruption to a given sentence (i.e., mask language model and replaced token detection). BERT-based Pre-trained Models are powerful techniques for generating sentence embeddings and are well-suited for language understanding tasks.

Table 1 Summarization of pre-trained models used in this paper, including the pre-training task, architecture, downstream tasks, and language type

Model	Training tasks	Architecture	Downstream tasks (SE tasks)	Language type
RoBERTa	MLM	EN	–	NL
BERT	MLM, NSP	EN	–	NL
ALBERT	MLM, NSP	EN	–	NL
BERTOverflow	MLM	EN	SER	NL
CodeBERT	MLM, RTD	EN	CR, CS QA, CT, BF, CSM	NL, PL
PLBART	DAE	ED	DD, CD, QA, CT, BF, CS, CG	NL, PL
CodeT5	SMLM, IT, IMLM, BDG	ED	DD, CD, QA, CT, BF, CSM, CG	NL, PL
CoTexT	SMLM	ED	DD, BF CSM, CG	NL, PL

Table 2 Description and abbreviation of the pre-training tasks mentioned in Table 1

Pre-train tasks	Abb	Description
Mask language modeling	MLM	Given an input where certain tokens are hidden (masked), the model predicts those missing tokens
Next sentence prediction	NSP	Assesses whether two provided sentences naturally appear in sequence
Replaced token detection	RTD	Recognizes whether a specific token in the input is artificially generated, rather than being from the original text
Denoising auto-encoding	DAE	From an input where tokens have been modified (through masking, deletion, or replacement), the model aims to reproduce the original input
Seq2Seq MLM	SMLM	Using an encoder-decoder architecture, the model tries to predict a sequence of tokens that have been masked out
Identifier tagging	IT	Classifies each token in the input based on whether it's an identifier or not
Identifier MLM	IMLM	In the context of code, this task involves predicting masked-out identifiers
Bimodal dual generation	BDG	Given a natural language description/code, it produces code/natural language description and vice versa

BERT BERT (Devlin et al. 2018) is based on the Transformer architecture (Vaswani et al. 2017) 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 (Zhu et al. 2015). 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 the masked positions. 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.

DistilBERT DistilBERT is a smaller and lighter version of the BERT model. As the number of parameters of PTMs is getting bigger and bigger, DistilBERT aims to reduce the computational cost of training and inference processes while preserving most of the performance of the larger models. In comparison with the BERT model, which has 12 hidden layers and 110 million parameters in total, DistilBERT only has six hidden layers and 66 million parameters. DistilBERT has leveraged knowledge distillation (Hinton et al. 2015) during the pre-training phase and shown that it is possible to retain 97% of BERT's language understanding abilities with a 40% reduction in the model size and 60% faster.

Table 3 Abbreviations of architecture type, langauge type, and downstream tasks in Table 1

Architecture type	Downstream tasks	
Encoder-only (EN)	Software Entity Recognition (SER)	Bug Fixing (BF)
Encoder-decoder (ED)	Code-to-Code Retrieval (CR)	Code Summarization (CSM)
Language Type	Code Search (CS)	Defect Detection (DD)
Natural Language (NL)	Code Question Answering (QA)	Clone Detection (CD)
Programming Language (PL)	Code Translation (CT)	Code Generation (CG)

RoBERTa RoBERTa (Liu et al. 2019b) 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 a larger batch size and learning rate on a dataset that is an order of magnitude larger than the training data of BERT (Devlin et al. 2018; Zhu et al. 2015).

DistilRoBERTa Similar to DistilBERT, DistilRoBERTa is the distilled version of the RoBERTa model. By leveraging the same pre-training strategy (i.e., knowledge distillation), DistilRoBERTa has 34% fewer parameters (i.e., 82 million parameters) than RoBERTa, but twice as fast.

ALBERT ALBERT (Lan et al. 2020b) is claimed as **A** **L**ite **BERT**. ALBERT involves two parameter reduction techniques: factorized embedding parameterization and cross-layer parameter sharing. Factorized embedding parameterization breaks down the large embedding matrix into two small matrices, and cross-layer parameter sharing prevents the parameter from growing with the depth of the network. Both techniques can effectively reduce the number of parameters without compromising the performance. 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 (Feng et al. 2020) is pre-trained on both natural language (NL) and programming language (PL) data from the CodeSearchNet database (Husain et al. 2020). 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 to understand both forms of data. The CodeBERT model has been proven practical in various SE-related downstream tasks, such as Natural Language Code Search (Zhou et al. 2021; Huang et al. 2021), program repair (Mashhadi and Hemmati 2021), etc (Feng et al. 2020).

Concurrent with the work by Feng et al., researchers from Huggingface have also released their own CodeBERT model,[5] which is also pre-trained with the CodeSearchNet data. The CodeBERT model released by Feng et al. has the same number of layers and parameters as RoBERTa (12 layers and 128 million parameters). In comparison, HuggingFace's model only has six layers and 84 million parameters. In our paper, we refer to the model trained by Feng et al. as the **CodeBERT** model, and the model trained by HuggingFace as the **CodeBERT-small** model.

BERTOverflow BERTOverflow (Tabassum et al. 2020) is a SE-domain PTM trained with 152 million sentences from Stack Overflow. The author of BERTOverflow introduces a software-related name entity recognizer (SoftNER) that combines an attention mechanism with code snippets. The model follows the same design as the BERT model with 110 million parameters. Experimental results showed that leveraging embedding generated by BERTOverflow in SoftNER substantially outperformed BERT in the code and named entity recognition task for the software engineering domain.

[5] https://huggingface.co/huggingface/CodeBERTa-small-v1

2.3.2 Encoder-decoder Pre-trained Models

In contrast with the BERT-based PTMs, which have an encoder-only architecture, encoder-decoder models follow the full transformer architecture.

CodeT5 Wang et al. proposed CodeT5 (Wang et al. 2021), which inherits the T5 (Text-To-Text Transfer Transformer) architecture and conducts the denoising sequence-to-sequence pre-training. To assist the model in better understanding programming languages, Wang et al. extend the denoising objective of T5 proposed with identifier tagging and prediction tasks. CodeT5 is trained to recognize which tokens are identifiers and to predict them from masked values. Moreover, CodeT5 leverages a bimodal dual-generation task for NL-PL alignment. CodeT5 provides the ability for multi-task learning, and it is capable of being fine-tuned in numerous downstream tasks, including code summarization, code generation, code translation, code refinement, defect prediction, clone detection, etc.

Wang et al. implemented several versions of CodeT5 with different sizes, where CodeT5-small has 60 million parameters and CodeT5-base has 220 million parameters. In this paper, we refer to CodeT5-base as **CodeT5** and CodeT5-small as **CodeT5-small**.

PLBART PLBART (Ahmad et al. 2021) is proposed by Ahmad et al. and is capable of performing a broad spectrum of program understanding and generation tasks. PLBART inherits the BART (Lewis et al. 2020) architecture and is trained using multilingual denoising tasks in Java, Python, and English. PLBART is pre-trained with a large-scale bi-model corpus of both natural languages and programming languages, including Java, Python, and English. PLBART has been shown to outperform the rival state-of-the-art methods in code summarization, code translation, and code generation tasks. It is also capable of generating promising performance in a wide range of language understanding tasks, e.g., program repair, clone detection, and vulnerable code detection.

CoTexT Phan et al. introduced CoTexT(**Co**de and **Tex**t Transfer **T**ransformer) (Phan et al. 2021), which is also implemented in the T5 architecture. In the pre-training stage, CoText mainly leverages the bi-model data collected from the CodeSearchNet corpus (Husain et al. 2020) and GitHub repositories. The effectiveness of CoTexT is demonstrated by evaluating four tasks, which are code summarization, code generation, defect detection, and code refinement. Results show that CoText is capable of achieving better performance than CodeBERT and PLBART.

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 Fig. 3. The three stages are: **Pre-processing, Feature Extraction, and Classification**. We first decompose each SO post into three components, i.e., *Title*, *Description*, and *Code*. Thus, *PTM4Tag+* is implemented with a *triplet* architecture and leverages three PTMs as encoders to generate representations for each component. Then, 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 concatenated to construct the final representation of the SO post. Finally, 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

During the pre-processing stage, we split an SO post into three components and then conduct tokenization.

3.1.1 Post Component Extraction

Figure 2 illustrates a typical SO post that consists of three components: *Title*, *Body*, and *Tags*. The *Title* summarizes the question, and the *Body* provides details of the question that helps readers to understand the question. Following prior studies (Baltes et al. 2018), we further divide the *Body* into *Description* blocks and *Code* blocks. *Description* blocks in a *Body* are narrative sections that describe the context or problem in natural languages. *Code* refers to the parts of the post that are enclosed in HTML tags `<pre><code>`. For simplicity, we refer to the sections enclosed in the `<pre><code>` HTML tags as *code snippets* in this paper. However, the sections enclosed in the `<pre><code>` HTML tags may not always be actual code snippets; they can also be other types of text, like stack traces or error messages.

Different from prior studies that discarded *Code* blocks in Post during the pre-processing stage (because the *Code* can be written by novices and have low quality), we keep *Code* in our work. This is because we observe that the *Code* blocks in an SO post can provide valuable semantic information in recommending tags. Take the SO post shown in Fig. 2 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 adding the *Code* into consideration, the grammar of Python can be easily used to infer that the post is likely to relate to the tag '`python`'. As a result, we consider that a post is made up of four components: 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*).

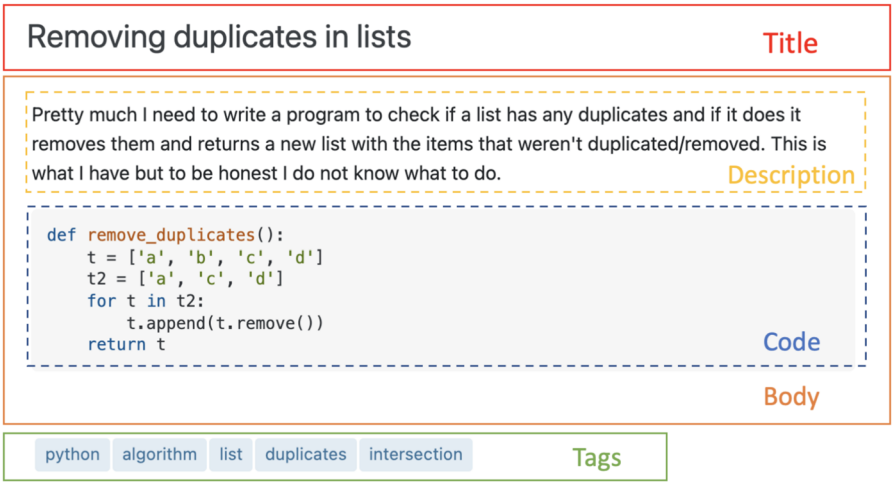

Fig. 2 An example of an SO Post. A post contains a short title that summarizes the main content of this post. The body of a post can include detailed descriptions written in natural languages and code snippets

To decompose the SO posts into the above-mentioned four constituents, we identify the *Title* and the *Tag* of posts from the `Posts.xml` in the official data dump released by the Stack Overflow website. To extract the *Code* in posts, we use a regular expression `<pre><code>([\s\S]*?)</code></pre>` to identify the code blocks in posts. 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.

3.1.2 Tokenization

Since the design of *PTM4Tag+* leverages transformer-based PTMs, we rely on the corresponding tokenizer of the underlying pre-trained model to generate token sequences. The conventional PTMs usually accept a maximum input sequence length of 512 sub-tokens, which also always include two special tokens $\langle CLS \rangle$ and $\langle SEP \rangle$. The $\langle CLS \rangle$ token (**CL**a**S**sification) is the first token of every input sequence, and the corresponding hidden state is then used as the aggregate sequence representation. $\langle SEP \rangle$ token (**SEP**arator) is inserted at the end of every input sequence. The problem of capturing long text arises since a significant proportion of the training samples has exceeded the maximum acceptable input limit of the PTMs.

By default, we tackle the problem using a *head-only* truncation strategy (Sun et al. 2020), which only considers the first 510 tokens (excluding the $\langle CLS \rangle$ and $\langle SEP \rangle$ tokens) as the input tokens. We also further discuss the impact of the *tail-only* truncation strategy in Section 5. Instead of filtering less significant words, we apply the truncation strategy as all the pre-trained models utilized in our study were not subjected to data filtering during their pre-training phase. Introducing significant changes to the input data structure may compromise the model's performance, as the altered input distribution might not align with the original data distribution the model was trained on.

3.2 Feature Extraction

During the feature extraction stage, we utilize PTMs as feature extractors to obtain representations for Title, Descripition, and Code components of a post. Different components convey information at varying degrees of detail and follow different text distributions. In our work, we consider Title, Description, and Code as three independent components and then leverage three PTMs to generate representations for each component. We then concatenate the representations of the three components as post-representation.

3.2.1 Language Modeling with PTMs

We use PTMs to generate embeddings for each component. Training transformer-based models from scratch can be extremely resource-intensive in terms of computational cost, time, and data requirements. Without adequate pre-training, the performance of such models can be considerably suboptimal. Taking the expensive cost at the pre-training stage into account, we leverage the released PTMs by the community in the design of *PTM4Tag+* to generate contextual word embeddings.

The pre-trained model part of *PTM4Tag+* is replaceable, and we have empirically implemented eight variants of *PTM4Tag+* with different PTMs (refer to Section 4.4) and investigated the impact of the PTM selection within *PTM4Tag+*.

3.2.2 Pooling and Concatenation

After the word embeddings are generated, we obtain the post representation by applying a pooling strategy. Pooling refers to the process of transforming a sequence of (token-level) word embeddings into sentence embedding (wherein a transformer-based model, each word embedding usually has 768 hidden dimensions). A pooling strategy executes the down-sampling on the input representation, and it condenses the granular (token-level) word embeddings into a fixed-length sentence embedding that is intended to capture the meaning of the whole context.

We consider two categories of PTMs in our work, i.e., BERT-based PTMs and encoder-decoder PTMs. We discuss how to get embeddings from these two types of PTMs respectively. Prior studies showed that there are several common choices to derive fixed-size sentence embeddings from BERT-based PTMs (Reimers and Gurevych 2019), including (1) using the first *CLS* token, (2) *Average Pooling* and (3) *Maximum Pooling*. More specifically, Reimers and Gurevych (Reimers and Gurevych 2019) have evaluated the effectiveness of different pooling strategies on the SNLI dataset (Bowman et al. 2015) and the Multi-Genre NLI dataset (Williams et al. 2018) 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.

While BERT-based PTMs are encoder-only, Encoder-decoder PTMs have the full transformer architecture with both encoder and decoder modules. According to Ni et al. (2022), typically, strategies to obtain sentence representations for encoder-decoder PTMs are (1) *Encoder-only first*: use the first token of the encoder output as the sentence embedding (2) *Encoder-only mean* use the average of the encoder outputs (3) *Encoder-Decoder first*: use the first decoder output as the sentence embedding. Experiments from Ni et al. showed that the *Encoder-only mean* produced the best performance. Hence, we adopt the second strategy to obtain component-wise embeddings for encode-decoder PTMs.

Finally, we concatenate the three component-wise embeddings (i.e., Title, Description, and Code) sequentially to obtain the final representation of an SO post.

3.3 Model Training and Inference

After performing average pooling, we concatenate the output embeddings and feed it into a feed-forward neural network to perform the task of multi-label classification. Given a training dataset consisting of $\mathcal{X}$ (a set of SO posts) and corresponding ground truth tags y for each $x \in \mathcal{X}$, we train a tag recommendation model f by minimizing the following objective:

$$\mathcal{L} = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{L} y_j \times \log(f(y_j|x_i)) + (1 - y_j) \times \log(1 - f(y_j|x_i)) \tag{1}$$

In the above equation, $N = |\mathcal{X}|$ is the total number of training examples and $f(y_j|x_i)$ is the probability that tag y_j is related to the 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 passes through both the multi-label classifier and the PTMs used to process *Title*, *Description*, and *Code*. The parameters of both the tag predictor 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 $f(y_j|x_i)$ in the vector corresponds to the probability that tag y_j 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 for a SO post.

4 Experimental Settings

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

4.1 Research Questions

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

RQ1: Out of the eight variants of *PTM4Tag+* with different PTMs, which gives the best performance?

Transformer-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 SQA sites. Past studies have shown that different PTMs have their own strengths and weaknesses. The effectiveness of PTMs varies by task due to the fact that they are pre-trained with various datasets, pre-training objectives, and vocabularies.

For example, Von der Mosel et al. (2022) found BERTOverflow outperforms the NLP-domain PTM, BERT, by a substantial margin in issue type prediction and commit intent prediction. Mosel et al. further reported that the vocabulary of SE-domain PTMs (i.e., BERTOverflow) contains many programming-related vocabularies such as *jvm, bugzilla, and debug* which are absent in the vocabulary of BERT (Devlin et al. 2018). However, Yang et al. reported that the NLP-domain PTMs (e.g. BERT and RoBERTa) perform better than SE-domain models, i.e., BERTOverflow in the API review classification task (Yang et al. 2022). Yang et al. claimed that this phenomenon may be because of the fact that the NLP-domain models are likely to be pre-trained on more extensive data. Thus, the efficacy of different pre-trained models on this task remains unclear. Moreover, in addition to the BERT-based PTMs which are encode-only, the encoder-decoder PTMs (e.g. CodeT5, PLBART, and CoText) have also performed surprisingly well in a wide spectrum of SE-related text understanding and generation tasks. It is intuitive to assume that the Encoder stacks of these models are capable of generating powerful text representation. However, not much SE literature has investigated the performance in classification tasks.

Since the underlying PTMs of *PTM4Tag+* are 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. Namely, we compare the results of NLP-domain BERT-based models (i.e., BERT Devlin et al. 2018, RoBERTa Liu et al. 2019b, ALBERT Lan et al. 2020b), SE-domain BERT-based models (i.e., Feng et al. 2020 and BERTOverflow Tabassum et al. 2020), and SE-domain Seq2Seq models (i.e., PLBART Ahmad et al. 2021, CoTexT Phan et al. 2021, and CodeT5 Wang et al. 2021).

RQ2: How is the performance of *PTM4Tag+* compared to the state-of-the-art approach in Stack Overflow tag recommendation?

The current state-of-the-art approach for recommending tags of SO posts is implemented based on a Convolutional Neural Network and trained from scratch (Xu et al. 2021). 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.

RQ3: Which component of post benefits *PTM4Tag+* the most?

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 their own perspective. Title summarizes the question with a few words; Description further expands the content from the Title; Code snippets often are a real example of the problem. Thus it motivates us to investigate which component produces the most critical contribution in the *PTM4Tag+* framework.

RQ4: How is the performance of *PTM4Tag+* with smaller PTMs?

The PTMs considered in RQ1 typically have 12 hidden layers with more than 100 million parameters. As our *PTM4Tag+* uses three PTMs to model different components of an SO post, the tool size is further increased by three times. Despite the encouraging performance, such a design has amplified the limitation of *PTM4Tag+* with respect to the inference latency. In the context of tagging SO posts, a faster inference speed could notably increase user satisfaction when running our tool. Therefore, it prompts our interest in experimenting with smaller PTMs under the *PTM4Tag+* framework. We adopted four additional smaller off-the-shelf PTMs. As introduced in Section 2, they are *DistilBERT*, *DistilRoBERTa*, *CodeBERT-small*, and *CodeT5-small*. To the best of our knowledge, we leveraged all available small variants of the PTMs from RQ1.

4.2 Data Preparation

To ensure a fair comparison to the current state-of-the-art approach (Xu et al. 2021), 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.[6] A tag is regarded as *rare* if its occurrence is less than a pre-defined threshold θ. 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 are less important and less useful to serve as representative tags to be recommended to users (Xia et al. 2013), which is a common practice in prior research (Xu et al. 2021; Xia et al. 2013; Zhou et al. 2019).

Following the same procedure as Xu et al., we set the threshold θ for deciding a rare tag as 50. We calculate the statistics for the occurrences of tags in the dataset. While the total occurrence of all tags in our dataset is 64,197,938, the sum of occurrence for tags occurring fewer than 50 times is 33,416. This represents just 0.05% of the total occurrences. Such a negligible percentage underlines the rarity and insignificance of these tags (tags that occur fewer than 50 times) within the entire dataset.

We remove all the rare tags of a post and the posts that contain rare tags only from the dataset. In the end, we have identified 29,357 rare tags and 23,687 common tags in total,

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

which is the same number in Xu et al.'s work. As a result, we obtained a dataset consisting of 10,379,014 posts. We selected 100,000 latest posts as the test data and used the rest of the 10,279,014 posts as the training data. Instead of random sampling, a chronological approach to splitting data is more representative of mimicking how the system works in real-world scenarios, especially for the Stack Overflow site.

4.3 Evaluation Metrics

Previous studies of tag recommendation (Xu et al. 2021; Li et al. 2020; Zhou et al. 2019) on SQA sites use $Precision@k$, $Recall@k$, $F1\text{-}score@k$ for evaluating the performance of the approaches. Following prior studies, given a corpus of SO posts, $\mathcal{X} = \{x_1, ...x_n\}$, we report $Precision@k_i$, $Recall@k_i$, $F1\text{-}score@k_i$ on each post x_i respectively where $0 \leq i \leq n$ and calculate the average of $Precision@k_i$, $Recall@k_i$, $F1\text{-}score@k_i$ as $Precision@k$, $Recall@k$, $F1\text{-}score@k$ 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 ith post in the test dataset, we denote its ground truth tags for a particular post by GT_i and predicted top-k tags of the model by Tag_i^k. We calculate $Precision@k_i$ as:

$$Precision@k_i = \frac{|GT_i \cap Tag_i^k|}{k} \tag{2}$$

Then we average all the values of $Precision@k_i$:

$$Precision@k = \frac{\sum_{i=1}^{|\mathcal{X}|} Precision@k_i}{|\mathcal{X}|} \tag{3}$$

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 (Xu et al. 2021; Zhou et al. 2019; Li et al. 2020), a modified version of $Recall@k$ is commonly adopted as indicated in (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 (Xu et al. 2021).

$$Recall@k_i = \begin{cases} \frac{|GT_i \cap Tag_i^k|}{k} & \text{if } |GT_i| > k \\ \frac{|GT_i \cap Tag_i^k|}{|GT_i|} & \text{if } |GT_i| \leq k \end{cases} \tag{4}$$

$$Recall@k = \frac{\sum_{i=1}^{|\mathcal{X}|} Recall@k_i}{|\mathcal{X}|} \tag{5}$$

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:

$$F1\text{-}score@k_i = 2 \times \frac{Precision@k_i \times Recall@k_i}{Precision@k_i + Recall@k_i} \tag{6}$$

$$F1\text{-}score@k = \frac{\sum_{i=1}^{|S|} F1\text{-}score@k_i}{|\mathcal{X}|} \tag{7}$$

Table 4 Variants of *PTM4Tag+*

Model name	BERT-Base model	Considered components	Architecture
BERT_{ALL}	BERT	Title, Description,Code	Triplet
RoBERTa_{ALL}	RoBERTa	Title, Description,Code	Triplet
ALBERT_{ALL}	ALBERT	Title, Description,Code	Triplet
CodeBERT_{ALL}	CodeBERT	Title, Description,Code	Triplet
$\text{BERTOverflow}_{ALL}$	BERTOverflow	Title, Description,Code	Triplet
CodeT5_{ALL}	CodeT5	Title, Description,Code	Triplet
PLBART_{ALL}	PLBART	Title, Description,Code	Triplet
CoTexT_{ALL}	CoText	Title, Description,Code	Triplet
$\text{CodeT5}_{NoTitle}$	CodeT5	Description,Code	Twin
CodeT5_{NoDesc}	CodeT5	Title, Code	Twin
CodeT5_{NoCode}	CodeT5	Title, Description	Twin
DistilBERT_{ALL}	DistilBERT	Title, Description,Code	Triplet
$\text{DistilRoBERTa}_{ALL}$	DistilRoBERTa	Title, Description,Code	Triplet
$\text{CodeBERT-small}_{ALL}$	CodeBERT-small	Title, Description,Code	Triplet
$\text{CodeT5-small}_{ALL}$	CodeT5-small	Title, Description,Code	Triplet

A large volume of literature on tag recommendation of SQA sites evaluates the result with k equals to 5, and 10 (Li et al. 2020). 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 $k \in \{1, 2, 3, 4, 5\}$.

For example, assume we have a Stack Overflow post with ground truth tags (i.e., python, machine-learning, neural-network, tensorflow, keras) and a set of predicted tags (i.e., python, pytorch, neural-network, tensorflow, and keras). We calculate the Precision@5 = 0.8 (1) and Recall@5 = 0.8 (3). Thus we can use these values to compute the F1-score@5 = 0.8 (5).

4.4 Implementation

To answer the four research questions mentioned in Section 1, we train fifteen variants of *PTM4Tag+*. Details about each variant model are summarized in Table 4.

For RQ1, we train eight variants of *PTM4Tag+* by using different PTMs (i.e., CodeBERT_{ALL}, ALBERT_{ALL}, BERT_{ALL}, RoBERTa_{ALL}, $\text{BERTOverflow}_{ALL}$, CodeT5_{ALL}, PLBART_{ALL}, CoTexT_{ALL} in Table 4) and empirically investigate their performance. Each variant follows the triplet architecture (as illustrated in Fig. 3).

For RQ2, we compare 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 reuse the replication package[7] released by the original authors.

In RQ3 we develop three ablated models, $\text{CodeT5}_{NoTitle}$, CodeT5_{NoDesc}, and CodeT5_{NoCode}, (as shown in Table 4) as CodeT5_{ALL} has the best performance from the experimental results of RQ1. Different to the variants from RQ1, each ablated model only contains two components and is implemented with a Twin architecture. In another words, two PTMs are leveraged in generating post embeddings, whereas the original design of *PTM4Tag+* involves three PTMs.

[7] https://github.com/maxxbw54/Post2Vec

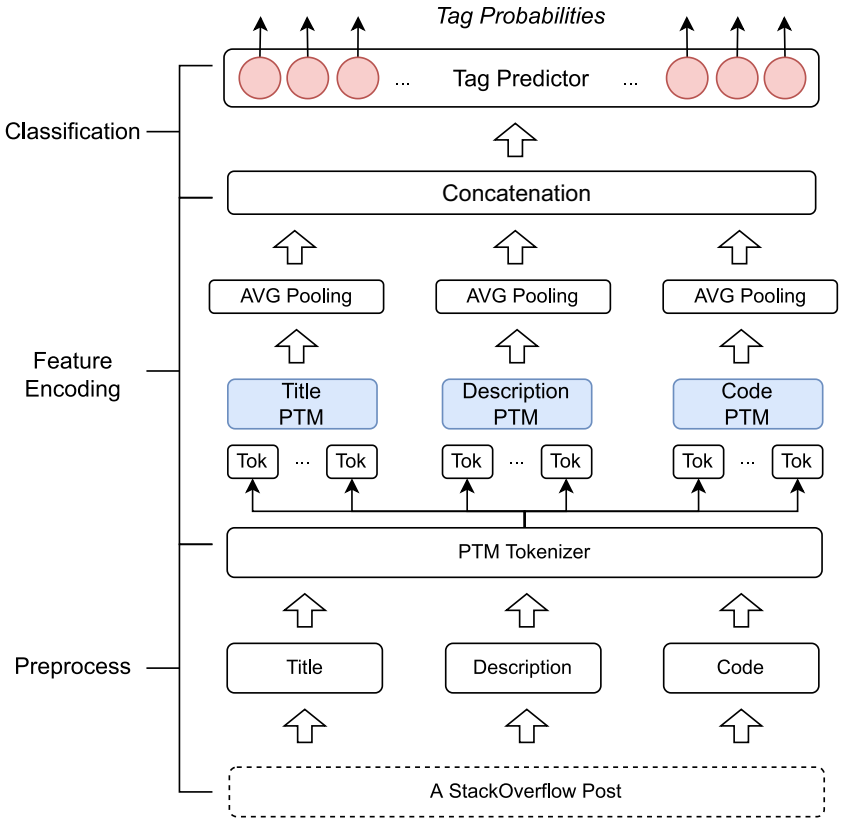

Fig. 3 The overview of the *PTM4Tag+* framework. The title, description, and code are extracted from an SO post and fed into three different pre-trained models to obtain embeddings for each of them. A classification model takes the processed embeddings as input and produces probabilities for each tag

The rest of the design is much similar, where we concatenate the component representation obtained from each encoder and train a multi-label classifier.

For RQ4, we additionally developed 4 variants of *PTM4Tag+* by using smaller PTMs (i.e., DistilBERT$_{ALL}$, DistilRoBERTa$_{ALL}$, CodeBERT-small$_{ALL}$, CodeT5-small$_{ALL}$ from Table 4). We compared the performance and inference latency of these models with the best-performing variant of *PTM4Tag+* from RQ1.

All the variants of *PTM4Tag+* are implemented with PyTorch V.1.10.0[8] and HuggingFace Transformer library V.4.12.3.[9] Considering the extensive amount of the data set, we only trained the models for one epoch at the fine-tuning stage. For each variant, we set the batch size as 64. We set the initial learning rate as 7E-05 and applied a linear scheduler to control the learning rate at run time.

[8] https://pytorch.org

[9] https://huggingface.co

5 Experimental Results

In this section, we report the experimental results of the variants under our proposed framework and the baseline approach. We further conduct an ablation study on our best-performing variant to assess the importance of the underlying component. Finally, we report the performance of *PTM4Tag+* on smaller PTMs with reduced model size. Based on the results, we answer the research questions presented in Section 4.

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

Results and Analysis To answer the question, we report the performance on eight variants of *PTM4Tag+* . We leverage three NLP-domain BERT-based PTMs (BERT, RoBERTa, and ALBERT), two SE-domain BERT-based PTMs (CodeBERT and BERTOverflow), and three SE-domain encoder-decoder PTMs (CodeT5, PLBART, and CoTexT). These variants are implemented with the Triplet architecture, which considers Title, Description, and Code as input. We refer to these variant models as BERT$_{ALL}$, RoBERTa$_{ALL}$, ALBERT$_{ALL}$, CodeBERT$_{ALL}$, BERTOverflow$_{ALL}$, CodeT5$_{ALL}$, PLBART$_{ALL}$, and CoText$_{ALL}$ respectively.

Table 5 illustrates the results of applying different PTMs in recommending the tags of SO posts and Fig. 4 draws the boxplot on the distribution of $F1\text{-}score@5$ for each variant. Performance-wise, CodeT5$_{ALL}$ achieves the highest rank and consistently outperforms the other variants in all evaluation metrics. For $F1\text{-}score@1\text{-}5$, it obtains the performance of 0.855, 0.726, 0.633, 0.568, and 0.519, respectively. ALBERT is the worst-performing model and BERTOverflow$_{ALL}$ is merely better than ALBERT$_{ALL}$.

Overall, BERTOverflow$_{ALL}$ and ALBERT$_{ALL}$ only achieve 0.427 and 0.406 in $F1\text{-}score@5$, which are lower than the other variants by a substantial margin. CodeBERT$_{ALL}$ and CoTexT$_{ALL}$ are tied for the second-best performing models, which both obtain an $F1\text{-}score@5$ of 0.513. In the next line, BERT$_{ALL}$ and RoBERTa$_{ALL}$ loseCodeBERT$_{ALL}$ and CoTexT$_{ALL}$ only by a small margin.

From Fig. 4, we can see that the 25% percentile value across most models is around 0.333 for $F1\text{-}score@5$. The median (50% percentile) value is a key metric for central tendency, and most models center around the 0.5 mark. However, CodeBERT$_{ALL}$, CoTexT$_{ALL}$, and CodeT5$_{ALL}$ slightly surpass this general trend, showcasing a median of more than 0.571.

As CodeT5$_{ALL}$, CoTexT$_{ALL}$, and PLBART$_{ALL}$ all achieve promising results, we demonstrate that encoder-decoder PTMs are also capable of generating good representations for SO posts. A possible reason for this is that these models adopt multi-task learning objectives at the training stage. Different to BERT-based PTMs, which follow the *pretrain-then-finetune* paradigm, encoder-decoder models like CodeT5 are trained with multiple tasks and multiple datasets at the same time (multi-task learning). Previous literature has claimed that multi-task learning leads to more generalized and better representations when being adapted to new tasks and domains (Liu et al. 2019a), our results conform to this claim. Moreover, CodeT5 uses more training data than CodeBERT and has more pre-training tasks than CoTexT. These may be attributed to its outstanding performance.

ALBERT$_{ALL}$ and BERTOverflow$_{ALL}$ are largely outperformed by the other variants. One possible reason for this could be that ALBERT has fewer parameters since its design aspires to address the GPU memory limitation. BERTOverflow follows the same architecture as BERT

Table 5 Comparison of all variants of *PTM4Tag+* with a triplet architecture and the baseline approach Post2Vec

Architecture	Domain	Model name	Precision@k				
			P@1	P@2	P@3	P@4	P@5
Encoder-only	SE	CodeBERT$_{ALL}$	0.848	0.701	0.579	0.486	0.415
		BERTOverflow$_{ALL}$	0.725	0.592	0.489	0.412	0.354
	NLP	RoBERTa$_{ALL}$	0.843	0.694	0.571	0.478	0.409
		BERT$_{ALL}$	0.845	0.696	0.575	0.482	0.413
		ALBERT$_{ALL}$	0.748	0.586	0.469	0.386	0.327
Encoder-decoder	SE	**CodeT5**$_{ALL}$	**0.855**	**0.708**	**0.586**	**0.492**	**0.420**
		PLBART$_{ALL}$	0.821	0.669	0.547	0.456	0.388
		CoTexT$_{ALL}$	0.848	0.701	0.579	0.486	0.415
–	–	Post2Vec	0.786	0.628	0.507	0.421	0.359
Architecture	Domain	Model Name	Recall@k				
			R@1	**R@2**	**R@3**	**R@4**	**R@5**
Encoder-only	SE	CodeBERT$_{ALL}$	0.848	0.756	0.724	0.733	0.757
		BERTOverflow$_{ALL}$	0.725	0.635	0.607	0.619	0.644
	NLP	RoBERTa$_{ALL}$	0.843	0.747	0.714	0.722	0.746
		BERT$_{ALL}$	0.845	0.750	0.719	0.728	0.752
		ALBERT$_{ALL}$	0.748	0.630	0.588	0.588	0.605
Encoder-decoder	SE	**CodeT5**$_{ALL}$	**0.855**	**0.763**	**0.732**	**0.741**	**0.765**
		PLBART$_{ALL}$	0.821	0.720	0.683	0.688	0.709
		CoTexT$_{ALL}$	0.848	0.755	0.723	0.733	0.756
–	–	Post2Vec	0.786	0.678	0.636	0.639	0.659
Architecture	Domain	Model Name	F1-score@k				
			F@1	**F@2**	**F@3**	**F@4**	**F@5**
Encoder-only	SE	CodeBERT$_{ALL}$	0.848	0.719	0.625	0.561	0.513
		BERTOverflow$_{ALL}$	0.725	0.606	0.527	0.475	0.427
	NLP	RoBERTa$_{ALL}$	0.843	0.711	0.617	0.553	0.505
		BERT$_{ALL}$	0.845	0.714	0.621	0.557	0.510
		ALBERT$_{ALL}$	0.748	0.600	0.506	0.447	0.406
Encoder-decoder	SE	**CodeT5**$_{ALL}$	**0.855**	**0.726**	**0.633**	**0.568**	**0.519**
		PLBART$_{ALL}$	0.821	0.686	0.590	0.526	0.480
		CoTexT$_{ALL}$	0.848	0.719	0.625	0.561	0.513
–	–	Post2Vec	0.786	0.646	0.549	0.488	0.445

and is designed with a vocabulary better suited to the software engineering domain (Von der Mosel et al. 2022), but it performs much worse than BERT$_{ALL}$ and RoBERTa$_{ALL}$ by a large margin. However, the experimental results suggested that BERTOverflow may still require additional training. It is potentially caused by the quality and size of the datasets used at the pre-training stage. BERTOverflow 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 can 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

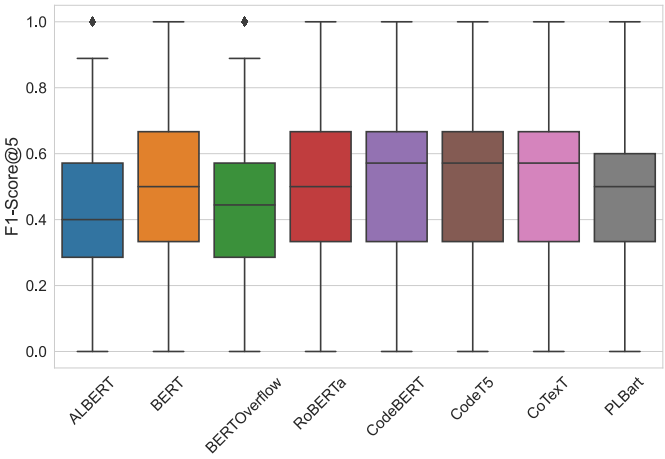

Fig. 4 Distribution of F1-Score@5 for All *PTM4Tag+* variants using a triplet architecture

sentence length of BookCorpus is 11 words. By estimation, the training corpus of BERT is likely to be twice more than BERTOverflow.

Moreover, although CodeBERT and BERTOverflow are both SE-domain PTMs, the performance of these two models is very different. This phenomenon could be because of that CodeBERT and BERTOverflow are initialized with different strategies before the pre-training starts. CodeBERT is initialized based on RoBERTa's checkpoint, whereas BERTOverflow is trained from scratch with SO data. Initializing with the checkpoint of another pre-trained model can inherit their knowledge and typically reduce the training effort by orders of magnitude (Rothe et al. 2020).

Comparing CodeBERT with BERT and RoBERTa, CodeBERT has utilized both natural language and programming language at the pre-training stage, while the other two models are trained solely with natural language data. As a result, CodeBERT is better at understanding programming languages and SE terminology. Given that many SO posts contain code snippets, this advantage allows CodeBERT$_{ALL}$ to achieve improved performance. In addition, the good performance of BERT and RoBERTa also suggests that pre-training models with a large scale of natural language data also be beneficial for programming language modeling.

Furthermore, conventional PTMs accept a maximum input sequence length of 512 subtokens. Our method utilizes a *head-only* truncation strategy by default, we further experiment with the effect of the *tail-only* truncation strategy. In Table 6, we present the performance of various variants of *PTM4Tag+*, in terms of $F1\text{-}score@k$ at different levels (k ranging from 1 to 5) with the tail-only truncation strategy. For convenience, we also add $F1\text{-}score@k$ of head-only truncation strategy in Table 6. When comparing the two truncation strategies for each variant, the performance differences are minimal. CodeT5$_{ALL}$ consistently showed the highest F1-scores across all values of k for both truncation strategies. Most variants display slightly better performance under the head-only truncation strategy as compared to the tail-only truncation. Notably, RoBERTa$_{ALL}$ shows a more significant drop in performance with tail-only truncation, especially for F1-score@1 and F1-score@2. This suggests that the head portion of the input might be more crucial for RoBERTa$_{ALL}$. CodeBERT$_{ALL}$, BERT$_{ALL}$, CodeT5$_{ALL}$, and CoTexT$_{ALL}$ show minimal variations in their performance across the two truncation strategies. The results presented emphasize the importance of PTM selection under

Table 6 Results of variants of *PTM4Tag+* for F1-score@k (k ranging from 1 to 5) with head-only and tail-only truncation strategies

Head-only truncation Model name	F@1	F@2	F@3	F@4	F@5
CodeBERT$_{ALL}$	0.848	0.719	0.625	0.561	0.513
BERT$_{ALL}$	0.845	0.714	0.621	0.557	0.510
RoBERTa$_{ALL}$	0.843	0.711	0.617	0.553	0.505
BERTOverflow$_{ALL}$	0.725	0.606	0.527	0.475	0.427
ALBERT$_{ALL}$	0.748	0.600	0.506	0.447	0.406
CodeT5$_{ALL}$	0.855	0.726	0.633	0.568	0.519
PLBART$_{ALL}$	0.821	0.686	0.590	0.526	0.480
CoTexT$_{ALL}$	0.848	0.719	0.625	0.561	0.513
Tail-only Truncation					
CodeBERT$_{ALL}$	0.847	0.718	0.624	0.560	0.512
BERT$_{ALL}$	0.844	0.714	0.620	0.556	0.509
RoBERTa$_{ALL}$	0.818	0.691	0.600	0.539	0.494
BERTOverflow$_{ALL}$	0.722	0.604	0.524	0.473	0.425
ALBERT$_{ALL}$	0.746	0.598	0.504	0.445	0.404
CodeT5$_{ALL}$	0.854	0.725	0.632	0.567	0.518
PLBART$_{ALL}$	0.821	0.685	0.590	0.525	0.479
CoTexT$_{ALL}$	0.847	0.718	0.624	0.560	0.512

the framework of *PTM4Tag+*, while the choice between head-only and tail-only truncation doesn't lead to vast performance differences. Given that the *head-only* truncation method has a slightly better $F1\text{-}score@k$, it has been selected as the default truncation strategy for subsequent experiments.

Answers to RQ1: Among the eight considered PTMs of *PTM4Tag+*, the one implemented with CodeT5 produces the best performance. Most PTMs from the SE domain give a more promising performance than PTMs from the NLP domain under *PTM4Tag+*.

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

Results and Analysis As presented in Table 5, the best performing variant of *PTM4Tag+*, i.e., CodeT5$_{ALL}$, substantially outperforms Post2Vec. In terms of $F1\text{-}score@k$ (where k ranges from 1 to 5), CodeT5$_{ALL}$ improved the performance by 8.8%, 12.4%, 15.3%, 16.4%, and 16.6%. On the other hand, CodeBERT$_{ALL}$ and CoTexT$_{ALL}$ surpass the performance of Post2Vec by 7.9%, 11.3%, 13.8%, 15.0% and 15.3%, respectively; BERT$_{ALL}$, RoBERTa$_{ALL}$, and PLBART$_{ALL}$ are also able to outperform Post2Vec by 4.4% to 14.6%. However, not all PTMs demonstrated exceptional performance under *PTM4Tag+*. Post2Vec outperformed BERTOverflow$_{ALL}$ by 8.4%, 6.6%, 8.5%, 2.7%, and 4.7%, and outperformed ALBERT$_{ALL}$ by 5.1%, 7.7%, 8.5%, 9.2%, and 9.6% in $F1\text{-}score@1$–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 give a better initialization of the model. Furthermore, CodeT5 provides in-domain knowledge of SE. Our results indicate that the knowledge learned in the pre-training stage is valuable to the success of the tag recommendation task.

Another potential reason for the superior performance of *PTM4Tag+* is that transformer-based models are more powerful than CNN in capturing long-range dependencies (Vaswani et al. 2017). The architecture of BERT-based PTMs is inherited from a Transformer. One of the critical concepts of Transformers is the *self-attention mechanism*, which enables its ability to capture long dependencies among all input sequences. Our results demonstrate the effectiveness and generalizability of transfer learning and reveal that the PTMs can achieve outstanding performance in the tag recommendation task for SO posts.

Answers to RQ2: CodeT5$_{ALL}$, PLBART$_{ALL}$, CoTexT$_{ALL}$, CodeBERT$_{ALL}$, BERT$_{ALL}$, and RoBERTa$_{ALL}$ outperform the state-of-the-art approach by a substantial margin. However, BERTOverflow$_{ALL}$ and ALBERT$_{ALL}$ demonstrated worse performance than the state-of-the-art approach.

RQ3. Which component of post benefits *PTM4Tag+* the most?

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., CodeT5.

The results for RQ3 are presented in Table 7. Notice that Table 7 also contains the results for the ablated model for CodeBERT$_{ALL}$, which is the best-performing variant in our previous ICPC paper. From the table, we identify that CodeT5$_{ALL}$ remains to be the best-performing model on all evaluation metrics. The results of both variants CodeT5$_{ALL}$ and CodeBERT$_{ALL}$ show that code plays the least important role. Excluding title and description also leads to a decline in all metrics, but the drop is less severe than when removing the code.

To provide a more intuitive understanding of the result, we further illustrate the performance gap of $F1\text{-}score@1$–5 between the ablated models and CodeT5$_{ALL}$ by visualizing in Fig. 5, where the value on the y axis is calculated using the score of CodeT5$_{ALL}$ minus the score of the ablated model. CodeT5$_{NoCode}$ 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 CodeT5$_{NoDesc}$ performed better in $F1\text{-}score@1$ and CodeT5$_{NoTitle}$ performed better in $F1\text{-}score@2$–5. It implies that Title is more important for boosting the performance of $F1\text{-}score@1$ and Description is more critical for improving $F1\text{-}score@2$–5. A possible explanation could be that Title always succinctly describes a post's central message, which could directly help the system predict the top tag. Description is usually much longer and elaborates the Title with more explanations; thus, it is more beneficial to recommend multiple tags. Moreover, as CodeT5$_{ALL}$ is still the best-performing model, it confirms that Code is a meaningful component and we need all three components in the tag recommendation task of SO.

Table 7 Experiment results of RQ3: Ablation study for post components using both CodeT5 and CodeBERT models

Model Name	Precision@k				
	P@1	P@2	P@3	P@4	P@5
CodeT5$_{\text{ALL}}$	**0.855**	**0.708**	**0.586**	**0.492**	**0.420**
CodeT5$_{NoCode}$	0.822	0.677	0.558	0.470	0.403
CodeT5$_{NoDesc}$	0.821	0.669	0.547	0.456	0.388
CodeT5$_{NoTitle}$	0.817	0.673	0.557	0.468	0.401
CodeBERT$_{\text{ALL}}$	**0.848**	**0.701**	**0.579**	**0.486**	**0.415**
CodeBERT$_{NoCode}$	0.823	0.682	0.562	0.472	0.408
CodeBERT$_{NoDesc}$	0.822	0.671	0.549	0.458	0.390
CodeBERT$_{NoTitle}$	0.808	0.664	0.547	0.460	0.394
Model Name	Recall@k				
	R@1	R@2	R@3	R@4	R@5
CodeT5$_{\text{ALL}}$	**0.855**	**0.763**	**0.732**	**0.741**	**0.765**
CodeT5$_{NoCode}$	0.822	0.728	0.697	0.707	0.732
CodeT5$_{NoDesc}$	0.821	0.720	0.684	0.689	0.710
CodeT5$_{NoTitle}$	0.817	0.724	0.695	0.705	0.730
CodeBERT$_{\text{ALL}}$	**0.848**	**0.756**	**0.724**	**0.733**	**0.757**
CodeBERT$_{NoCode}$	0.823	0.733	0.702	0.712	0.737
CodeBERT$_{NoDesc}$	0.822	0.723	0.686	0.693	0.714
CodeBERT$_{NoTitle}$	0.808	0.715	0.683	0.693	0.718
Model Name	F1-score@k				
	F@1	F@2	F@3	F@4	F@5
CodeT5$_{\text{ALL}}$	**0.855**	**0.726**	**0.633**	**0.568**	**0.519**
CodeT5$_{NoCode}$	0.822	0.694	0.603	0.543	0.499
CodeT5$_{NoDesc}$	0.821	0.686	0.591	0.527	0.480
CodeT5$_{NoTitle}$	0.817	0.690	0.600	0.541	0.496
CodeBERT$_{\text{ALL}}$	**0.848**	**0.719**	**0.625**	**0.561**	**0.513**
CodeBERT$_{NoCode}$	0.823	0.699	0.607	0.545	0.500
CodeBERT$_{NoDesc}$	0.822	0.688	0.593	0.530	0.483
CodeBERT$_{NoTitle}$	0.808	0.680	0.591	0.531	0.487

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-1 most relevant tags, and the contribution of Description increases when the number of predicted tags increases from two. Still, considering all three components achieve the best performance.

RQ4. How is the performance of *PTM4Tag+* with smaller PTMs?

Results and Analysis The performance of smaller PTMs under the *PTM4Tag+* framework is summarized in Table 8. DistilBERT gives the best performance. The worst-performing variant of smaller PTMs is CodeT5-small$_{ALL}$, where it achieves scores of 0.824, 0.689, 0.593, 0.529, and 0.482 with respect to $F1\text{-}score@1\text{-}5$. We also observe that these smaller

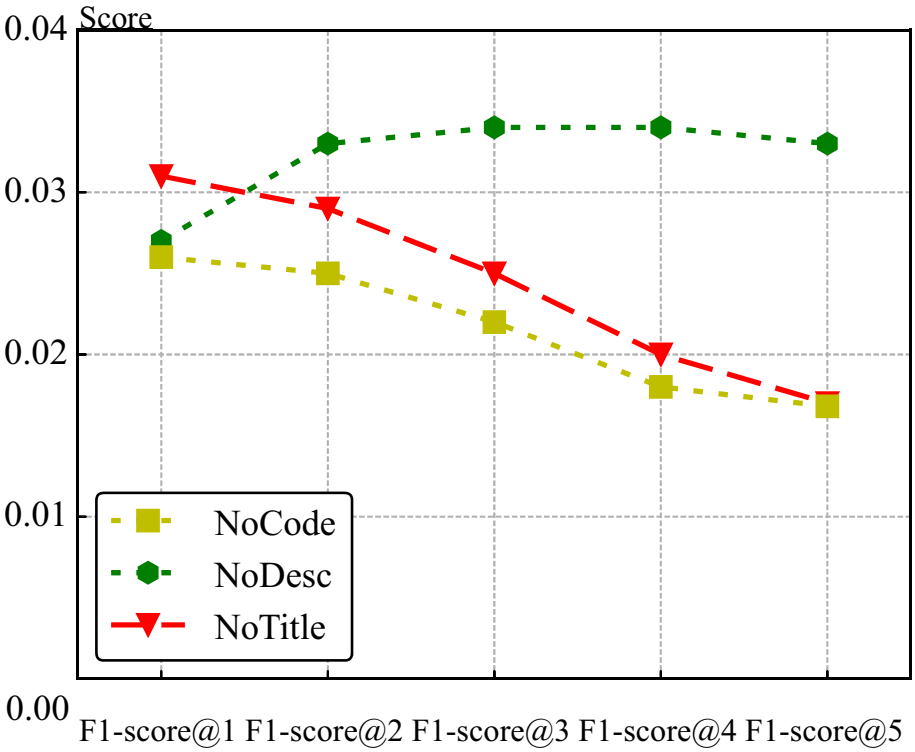

Fig. 5 A line chart demonstrate performance difference in $F1\text{-}score@k$ between each ablated models and CodeT5$_{ALL}$, where $k \in \{1, 2, 3, 4, 5\}$. The value on the y-axis is calculated using the corresponding score of the candidate ablated model minus the corresponding score of CodeT5$_{ALL}$

variants all can outperform the previous state-of-the-art method, Post2vVec, by a substantial margin.

Inference latency refers to the time taken for each model to make predictions. Generally speaking, inference latency is affected by the computing power of the running machine and the length of the input sequence. To make a fair comparison, we adopt the same hardware to query the models. To be specific, we use two Nvidia Tesla v100 16GB GPUs to run the model. As specified in Section 4, we set the input lengths to be the same as in training, which is 100 for Title, 512 for Description, and 512 for Code. We randomly sample 2,000 examples from the test set and calculate the average inference latency taken by the models. To further reduce the effects of randomness, the experiments are repeated five times. s

Table 9 summarizes the inference time improvement and performance drop of smaller PTMs compared with CodeT5 under the *PTM4Tag+* framework. On average, the inference latency is reduced by over 47.2% while at least 93.96% of the original performance could be preserved in terms of average $F1\text{-}score@k$. In Table 10, we present a detailed statistical summary of the inference time (measured in milliseconds) for CodeT5$_{ALL}$ and several smaller variants of *PTM4Tag+* , based on a sample size of 10,000 (2,000 $\times$ 5). Figure 6 presents the boxplot on the distribution of inference time. The CodeT5-base model exhibited the highest average inference time at 37.8 ms. DistilBERT and DistilRoBERTa demonstrated more moderate average inference times of 18.6 ms and 20.0 ms, respectively. The CodeBERT-small and CodeT5-small models yielded similar results, with mean inference times of 19.5 ms and 19.1

Table 8 Comparison of variants of *PTM4Tag+* with smaller pre-trained models and the best-performing variant of *PTM4Tag+*

Model name	Precision@k				
	P@1	P@2	P@3	P@4	P@5
CodeT5$_{ALL}$	**0.855**	**0.708**	**0.586**	**0.492**	**0.420**
DistilBERT$_{ALL}$	0.835	0.684	0.560	0.468	0.399
DistilRoBERTa$_{ALL}$	0.830	0.679	0.557	0.464	0.396
CodeBERT-small$_{ALL}$	0.831	0.681	0.559	0.467	0.398
CodeT5-small$_{ALL}$	0.824	0.672	0.549	0.457	0.390
Model Name	Recall@k				
	R@1	R@2	R@3	R@4	R@5
CodeT5$_{ALL}$	**0.855**	**0.763**	**0.732**	**0.741**	**0.765**
DistilBERT$_{ALL}$	0.835	0.737	0.701	0.707	0.729
DistilRoBERTa$_{ALL}$	0.830	0.731	0.695	0.701	0.723
CodeBERT-small$_{ALL}$	0.831	0.733	0.699	0.705	0.727
CodeT5-small$_{ALL}$	0.824	0.723	0.686	0.691	0.711
Model Name	F1-score@k				
	F@1	F@2	F@3	F@4	F@5
CodeT5$_{ALL}$	**0.855**	**0.726**	**0.633**	**0.568**	**0.519**
DistilBERT$_{ALL}$	0.835	0.702	0.605	0.541	0.493
DistilRoBERTa$_{ALL}$	0.830	0.696	0.601	0.536	0.489
CodeBERT-small$_{ALL}$	0.831	0.698	0.604	0.540	0.492
CodeT5-small$_{ALL}$	0.824	0.689	0.593	0.529	0.482

ms, respectively. From Table 10, we can clearly see that CodeT5$_{ALL}$ has the highest inference time, with a median of 37.9ms and a maximum of 114.7ms. In contrast, the smaller models, including DistilBERT, DistilRoBERTa, CodeBERT-small, and CodeT5-small, have notably faster inference times, with medians ranging from 18.5ms to 19.7ms. Among these, CodeT5-small has the shortest minimum inference time at 16.9ms. The standard deviation values suggest that the inference times for these models are relatively consistent, with CodeT5-base having the most variability.

Our results demonstrate that smaller variants of *PTM4Tag+* could outperform larger models like BERTOverflow$_{ALL}$, PLBART$_{ALL}$. Such a phenomenon suggests the performance of a model does not increase linearly with its size in the SO post-tagging task and it implies parameter redundancy of large models. Much previous literature showed that smaller PTMs could also give competent performance and the performance gap to larger PTMs is insignificant (Adoma et al. 2020; Giorgi et al. 2020) and sometimes they are even better when the training is carefully conducted (Sarfraz et al. 2021). For example, Wang et al. claimed that CodeT5-small yields better performance than PLBART with a smaller size in tasks like code summarization and code generation (Wang et al. 2021). For a practical tool, the trade-off between inference latency and performance should be cautiously determined.

Answers to RQ4: Using smaller PTMs under *PTM4Tag+* , the inference latency is reduced by over 47.2% on average while at least 93.96% of the original performance could be preserved in terms of average $F1\text{-}score@k$.

Table 9 Comparison of smaller PTMs under *PTM4Tag+* framework with the best-performing model, CodeT5$_{ALL}$, including the mean inference time(ms), inference time improvement(%), and F1-Score Performance drop(%)

Model name	Inference latency(ms)	F1-Score performance drop				
		F1@1	F1@2	F1@3	F1@4	F1@5
DistilBERT$_{ALL}$	18.6(−50.9%)	−2.12%	−3.20%	−3.84%	−4.46%	−4.68%
DistilRoBERTa$_{ALL}$	20.0(−47.2%)	−1.53%	−2.36%	−3.2%	−3.57%	−3.90%
CodeBERT-small$_{ALL}$	19.5(−48.5%)	−2.00%	−2.92%	−3.36%	−3.74%	−4.09%
CodeT5-small$_{ALL}$	19.1(−49.6%)	−2.83%	−4.17%	−5.12%	−5.70%	−6.04%
CodeT5$_{ALL}$	**37.8**	–	–	–	–	–

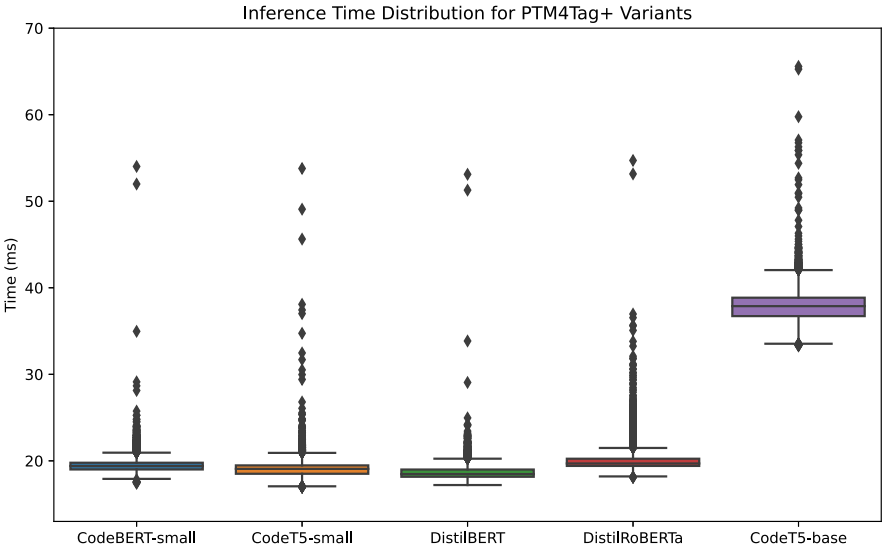

Fig. 6 A box-plot demonstrate the distribution of inference time of CodeT5$_{ALL}$ and small variants of *PTM4Tag+* among 10,000 samples

Table 10 Statistical summary for distribution of inference time (ms) of CodeT5$_{ALL}$ and small variants of *PTM4Tag+* among 10,000 samples

Model name	std	min	25%	50%	75%	max
CodeT5-base	2.0	33.3	36.7	37.9	38.9	114.7
DistilBERT	0.8	17.2	18.2	18.5	19.0	53.1
DistilRoBERTa	1.4	18.1	19.4	19.7	20.3	54.7
CodeBERT-small	0.9	17.4	19.0	19.4	19.8	54.0
CodeT5-small	1.1	16.9	18.5	19.1	19.5	53.8

6 Discussion

6.1 Error Analysis

We conduct an error analysis to illustrate the capability of our proposed framework *PTM4Tag+*. Take a Stack Overflow post[10] 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 (Husain et al. 2020) which is used to pre-train CodeT5 includes source code files that relate to both *memoization* and *decorator*.[11] This potentially could indicate that the pre-trained knowledge learned by CodeT5 is beneficial for our task.

From the 100,000 posts in the test data, 2,707 posts yield an $F1\text{-}score@5$ of zero. To comprehend which tags are less effectively recognized by *PTM4Tag+*, we count the occurrences of ground truth tags associated with posts that received an $F1\text{-}score@5$ of zero. Table 11 presents the top-10 most often missed tags of this analysis, showing that *python* and *python-3.x* are the tags most frequently missed by *PTM4Tag+*. We notice that this is because *PTM4Tag+* cannot accurately handle the version of python; it may likely produce the prediction of *python-2.7*.

6.2 Manual Evaluation

Furthermore, we conducted a manual evaluation of the predicted tags of *PTM4Tag+*. We randomly sample a statistically representative subset of 166 posts in our testing dataset which gives us a confidence level of 99% with a confidence interval of 10.[12] We invite three software developers with at least 5 years of programming experience to evaluate the relevancy of the tags predicted by *PTM4Tag+*. For each post, we present the top-5 tags predicted by *PTM4Tag+*. Each of the developers is required to annotate 166 posts individually, thus 166 x 5 = 830 tags in total. Each annotator is required to use domain expertise and is allowed to refer to external sources when checking the relevance of the predicted tags. Out of the 166 posts, the tags predicted for 76 posts are marked as containing no irrelevant tags by all three annotators. We then asked the three annotators to discuss their evaluation results. After the discussion, the three annotators agreed that there were a total of 49 posts containing 62 irrelevant tags.

From our manual analysis, we observed that a significant majority of the predicted tags are related to the post. However, some inaccuracies were also noted. We found that most of the irrelevant predictions of *PTM4Tag+* are not completely wrong. Instead, these irrelevant predictions are indirectly related to the topic of the post. We present six examples in Table 11. For example, for post 52109809, the predicted tags are: `bitbucket`, `git`, `git-config`, `github`, `ssh`. However, the post is solely about `git` and is not related to `bitbucket`,

[10] https://stackoverflow.com/questions/51910978

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

[12] we use the sample size calculator from https://www.surveysystem.com/sscalc.htm

Table 11 Top-10 tags most often missed in posts with an $F1\text{-}score@5$ of 0

	Frequency
Python-3.x	58
Python	45
Javascript	37
c#	29
Java	29
php	22
Angular	22
Android	20
Flutter	18
Web	18
.net	17

although `bitbucket` and `github` are related topics. Another noticeable observation is the simultaneous predictions of tags such `python`, `python-3.x`, and `python 2.7` always come together, which also outlines a limitation of our approach.

Further, we found the *PTM4Tag+* yielded wrong predictions in certain niche topics. For example, for post 51903596, *PTM4Tag+* gives the prediction of `ruby` and `julia-lang`, while the question is about the crystal programming language (Table 12).

6.3 Revised Experiments for Updated Dataset

To further assess the efficacy of *PTM4Tag+*, we conduct the evaluation of *PTM4Tag+* on an augmented dataset sourced from the most recent Stack Overflow dump dated June 2023.[13] Following the preprocessing procedures we mentioned in Section 4, tags with occurrences fewer than 50 times were filtered out. This updated dataset comprises 23,687 common tags and 22,498,254 posts. We use 22,398,254 posts as the training set and the latest 100,000 posts as the test set. We train CodeT5$_{ALL}$ on the updated dataset, which is the variant that has the highest mean F1-score@5 on the previous dataset. The experiment is conducted in the same setting we mentioned in Section 4. The experimental result is demonstrated in Table 13. Overall, CodeT5$_{ALL}$ yields an $F1\text{-}score@5$ of 0.516 on the latest data, which is comparable to 0.519 from the previous dataset.

6.4 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.[14] To instantiate variants of *PTM4Tag+* with different pre-trained models, we utilized a widely-used deep learning library *Hugging Face*.[15] Similar to prior studies (Zhou et al. 2019; Wang et al. 2015; Xu et al. 2021), our work assumes that the tags are labeled correctly by users in Stack Overflow. However, some tags are potentially

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

[14] https://github.com/maxxbw54/Post2Vec

[15] https://huggingface.co/

Table 12 Examples of predicted tags of PTM4Tag+

Post ID	Title	Predicted tags
51903596	How to resolve generics in macros?	Elixir, julia-lang, macros, metaprogramming, ruby
52109809	Same user set after changing username globally to git	Bitbucket, git, git-config, github, ssh
52018972	Using else in basic program getting error: unindent does not match any outer indentation level	If-statement, indentation, python, python−2.7, python-3.x
51929049	After running the above code why i am getting just one output for type () function? why not for all three?	Function, python, python−2.7, python-3.x, types
51905926	Python how to create a string that is 4 characters long from a number	Integer, python, python−2.7, python-3.x, string

Table 13 Experiment results for CodeT5$_{ALL}$ on the updated dataset

Model name	Precision@k				
	P@1	P@2	P@3	P@4	P@5
CodeT5$_{ALL}$	0.839	0.703	0.585	0.494	0.424
Model Name	Recall@k				
	R@1	R@2	R@3	R@4	R@5
CodeT5$_{ALL}$	0.839	0.752	0.716	0.722	0.744
Model name	F1-score@k				
	F@1	F@2	F@3	F@4	F@5
CodeT5$_{ALL}$	0.839	0.719	0.627	0.563	0.516

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 4.2). We conducted a manual analysis to assess the relevancy of the tags of Stack Overflow posts. We computed a statistically representative sample size using a popular sample size calculator[16] with a confidence level of 99% and a confidence interval of 10. We sampled 166 code snippets to conduct the manual evaluation where three experienced developers with at least 5 years of programming background were involved in this manual evaluation. They individually assessed the relevance of the tags, leveraging both their expertise and external sources. A discussion was then held among the three evaluators if they identified any conflicts. They all agreed that the tags associated with the sampled posts were relevant to the content. Another threat to the internal validity is the hyperparameter setting we used to fine-tune *PTM4Tag+*. To mitigate this threat, we use hyper-parameters that were reported in prior studies as recommended or optimal (Devlin et al. 2018; Feng et al. 2020; Wang et al. 2022a).

Users of SO can put any kind of text into the *Code* blocks of SO posts. The *Code* component of *PTM4Tag+* may contain other types of content than code snippets, such as stack traces and error messages. We refer to these content as non-code content. To study the impact of non-code contents on our framework, we randomly sample a statistically representative subset of 166 posts from our test dataset, providing us with a 99% confidence level and a 10% confidence interval. On manual inspection of this subset, we discern that 36 posts, which is 21.7% of the subset, have content other than typical code snippets within the *Code* component. Breaking it down, 22 posts featured error messages, 5 posts detailed database schema layouts, 5 posts delineated input/output format descriptions, and 4 posts showcased actual program outputs. The performance of *PTM4Tag+* (CodeT5$_{ALL}$) in this sampled subset and the posts containing non-code content are presented in Table 14. Notably, the average $F1\text{-}score@5$ for posts with non-code content surpasses that of the overall subset. This suggests that *PTM4Tag+* has the potential to not only deal with typical code snippets but also with other forms of content within the *Code* component.

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.

[16] https://www.surveysystem.com/sscalc.htm

Table 14 Comparison of *PTM4Tag+* average performance on a statistically representative subset from the test set versus *PTM4Tag+* average performance on Non-code Text within the same subset

	P@1	P@2	P@3	P@4	P@5
Subset	0.867	0.732	0.604	0.503	0.435
Non-code	0.886	0.771	0.629	0.521	0.446
	R@1	**R@2**	**R@3**	**R@4**	**R@5**
Subset	0.867	0.792	0.742	0.735	0.769
Non-code	0.886	0.786	0.705	0.698	0.720
	F@1	**F@2**	**F@3**	**F@4**	**F@5**
Subset	0.867	0.752	0.647	0.573	0.530
Non-code	0.886	0.776	0.656	0.583	0.535

P represents Precision, R represents Recall, and F represents the F1-Score

Threats to Construct Validity

Threats to construct validity are related to the suitability of our evaluation metrics. $Precision@k$, $Recall@k$, and $F1\text{-}score@k$ are widely used to evaluate many tag recommendation approaches in software engineering (Zhou et al. 2019; Wang et al. 2015; Xu et al. 2021). Thus, we believe the threat is minimal. We reuse the evaluation metrics proposed in our baseline method, Post2Vec (Xu et al. 2021). We conduct the Wilcoxon signed-rank statistical hypothesis test (Gehan 1965) on the paired data which corresponds to the F1-score@5 of CodeT5$_{ALL}$ and all other PTM models. We conducted the Wilcoxon Signed Rank Test at a 95% confidence level (i.e., p-value < 0.05). We found that CodeT5$_{ALL}$ significantly outperforms all other variants with a threshold of $p < 0.05$. In addition, we conducted Cliff's delta (Cliff 1993) to measure the effect size of our results. The Cliff's Delta statistic, denoted as $|\delta|$, is a non-parametric effect size measure that quantifies the amount of difference between two groups of observations beyond p-values interpretation. We consider $|\delta|$ that are classified as "Negligible (N)" for $|\delta| < 0.147$, "Small (S)" for $0.147 \leq |\delta| < 0.33$, "Medium (M)" for $0.33 \leq |\delta| < 0.474$, and "Large (L)" for $|\delta| \geq 0.474$, respectively following previous literature (Cliff 2014). We observe that CodeT5$_{ALL}$ substantially outperforms the baseline models - Cliff's deltas are not negligible and are in the range of 0.21 (small) to 0.54 (large).

6.5 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 (Li et al. 2020; Zhou et al. 2019; Wang et al. 2015; Xu et al. 2021). Researchers have tackled the problem via a range of techniques, e.g., collaborative filtering (Li et al. 2020) and deep learning (Xu et al. 2021). Furthermore, these techniques usually involve separate training for each component. Our experiment 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.

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. 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. 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

Encoder-decoder models should also be considered in SO-related tasks for classification tasks. By convention, BERT-based models are widely used for classification tasks in generating sentence embeddings. Our results have shown that encoder-decoder models are also capable of generating meaningful embeddings (especially CodeT5 gives the best performance) in the tag recommendation task of SO posts. We advocate researchers also involve the encoder-decoder models as baseline methods for SO-related classification tasks in the future.

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 (Zhou et al. 2019; Li et al. 2020; Wang et al. 2015; Zhou et al. 2017). However, our results show that code snippets are also beneficial to capturing the semantics of SO posts and further boosting the performance of the tag recommendation task. We encourage researchers to consider both the natural and programming language parts of a post when analyzing SQA sites.

Lesson #4

Smaller pre-trained models are practical substitutes. We demonstrate that various small PTMs could achieve similar performance to larger PTMs while increasing the inference latency. Smaller variants of *PTM4Tag+* even outperformed variants with BERTOverflow and PLBART. We show that smaller PTMs are also effective in the considered task, and developers should consider these PTMs to reach a balance point between the performance and usability in the real world for SO-related tasks.

7 Related Work

7.1 Pre-trained Models

Transformer-based pre-trained models have recently benefited a broad range of both understanding and generation tasks. Recent works (Lee et al. 2019; Beltagy et al. 2019; Huang et al. 2020) have shown that the in-domain knowledge acquired by PTMs is valuable in improving the performance on domain-specific tasks, such as ClinicalBERT (Huang et al. 2020) for clinical text, SciBERT (Beltagy et al. 2019) for scientific text, and BioBERT (Lee et al. 2019) for biomedical text. Evoked by the success of PTMs in other domains, researchers

have started to work on the SE domain-specific PTMs (Feng et al. 2020; Shi et al. 2024; Zhou et al. 2023a; Tabassum et al. 2020). Developers are free to create and pick arbitrary identifiers. These identifiers introduce a lot of customized words into the texts within the SE field (Shi et al. 2022). This suggests that models pre-trained on general texts, such as BERT (Devlin et al. 2018), may not be optimal for representing texts in the software engineering domain.

Specifically, the transformer model is designed with an encoder-decoder architecture. The encoder takes an input sentence and derives important features from it, while the decoder leverages these features to generate an output sentence. In terms of architecture, we categorize the transformer-based pre-trained models into three types, which are encoder-only models, decoder-only models, and encoder-decoder models

Encoder-only models are widely generating sentence representations in language understanding tasks (Lan et al. 2020a). Buratti et al. trained C-BERT (Buratti et al. 2020) using code from the top-100 starred GitHub C language repositories. Experimental results show that C-BERT achieves high accuracy in the Abstract Syntax Tree (AST) tagging task and produces comparatively good performance to graph-based approaches on the software vulnerability identification task. Von der Mosel et al. (2022) aims to provide a better SE domain-specific pre-trained model than BERTOverflow (Tabassum et al. 2020) and propose seBERT (Von der Mosel et al. 2022). He et al. pre-trained SOBERT (He et al. 2024) based on the corpus of Stack Overflow posts. CCBERT (Zhou et al. 2023b) learns a generic representation of code changes. It perceives fine-grained code changes at the token level. Guo et al. presented GraphCodeBERT (Guo et al. 2021), the first pre-trained models that consider the inherent structure of programming languages. In addition to NL and PL data, GraphCodeBERT also involves data flow information. In the pre-training stage, Guo et al. utilize masked language modeling and two new structure-aware tasks as pre-training objectives. TreeBERT is a tree-based pre-trained model for improving code-related generation tasks, proposed by Jiang et al. (2021). TreeBERT leverages the abstract syntax tree (shortened as AST) corresponding to the code into consideration. The model is pre-trained by the tree-masked language modeling (TMLM) task and node order prediction (NOP) task. As a result, TreeBERT achieved state-of-the-art performance in code summarization and code documentation tasks.

Decoder-only pre-trained models only inherit the decoder part of the transformer architecture. Decoder-only models are usually used for generation tasks (Wang et al. 2022b). Svyatkovskiy et al. trained GPT-C (Svyatkovskiy et al. 2020), a generative model based on GPT-2 architecture, and leveraged GPT-C to build a code completion tool.

Encoder-decoder pre-trained models use complete Transformer architecture. PLBART (Ahmad et al. 2021) has undergone pre-training on a massive set of Java and Python functions and corresponding natural language text through the denoising autoencoding process. Results showed that PLBART is promising in code summarization, code generation, and code translation tasks. Wang et al. introduced CodeT5 (Wang et al. 2021), a pre-trained encoder-decoder Transformer model that effectively utilizes the semantic information conveyed through developer-assigned identifiers. Extensive experiments demonstrate that CodeT5 significantly outperforms previous methods in tasks such as code defect detection and clone detection, as well as generation tasks including PL-NL, NL-PL, and PL-PL translations. CoTexT also adopted the T5 architecture (Phan et al. 2021). CoTexT is another bi-model PTM and is capable of supporting a range of natural language-to-programming language tasks, including code summarization and documentation, code generation, defect detection, and debugging.

7.2 Tag Recommendation for SQA Sites

Researchers have extensively studied the tag recommendation task in the SE domain and proposed a number of approaches. Wang et al. (2015) proposed TagCombine, a tag recommendation framework that consists of three components: a multi-label ranking component, a similarity-based ranking component, and a tag-term-based ranking component. Eventually, TagCombine leverages a sample-based method to combine the scores from the three components linearly as the final score. Wang et al. introduced EnTagRec (Wang et al. 2014) 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 extend EnTagRec (Wang et al. 2014) to EnTagRec++ (Wang et al. 2018), the latter of which additionally considers user information and an initial set of tags provided by a user. Zhou et al. proposed TagMulRec (Zhou et al. 2017), a collaborative filtering method that suggests new tags for a post based on the results of semantically similar posts. Li et al. proposed TagDC (Li et al. 2020), which is implemented with two parts: TagDC-DL which leverages a content-based approach to learn a multi-label classifier with a CNN Capsule network, and TagDC-CF which utilizes collaborative filtering to focus on the tags of similar historical posts. Post2Vec (Xu et al. 2021) distributively represents SO posts and is shown to be useful in tackling numerous Stack Overflow-related downstream tasks.

8 Conclusion and Future Work

In this work, we introduce *PTM4Tag+*, a pre-trained model-based framework for tag recommendation of Stack Overflow posts. We implement eight variants of *PTM4Tag+* with different PTMs. Our experiment results show that the CodeT5 gives the best performance under the framework of *PTM4Tag+*, and it outperforms the state-of-the-art approach by a large margin in terms of $F1\text{-}score@5$. However, *PTM4Tag+* variants implemented with BERTOverflow and ALBERT do not give promising results. Even though PTMs are shown to be powerful and effective, PTMs behave differently, and the selection of PTMs needs to be carefully decided.

In the future, we are interested in applying *PTM4Tag+* on more SQA sites such as AskUbuntu,[17] etc., to evaluate its effectiveness and generalizability further. It is noteworthy that beyond actual code snippets, SO posts often incorporate other textual artifacts like stack traces and error messages. We plan to expand our approach to more fine-grained types of text for future research.

Acknowledgements 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.

Data Availability We release our replication package to facilitate future research. The code and datasets used during the study are available in the https://doi.org/10.6084/m9.figshare.24268831.

Declarations

[17] https://askubuntu.com/

Conflict of interest The authors have no Conflict of interest/Conflict of interest to declare that are relevant to this paper.

References

Adoma AF, Henry NM, Chen W (2020) Comparative analyses of bert, roberta, distilbert, and xlnet for text-based emotion recognition. In: 2020 17th International Computer Conference on Wavelet Active Media Technology and Information Processing (ICCWAMTIP), IEEE, pp 117–121

Ahmad WU, Chakraborty S, Ray B, Chang K (2021) Unified pre-training for program understanding and generation. In: Toutanova K, Rumshisky A, Zettlemoyer L, Hakkani-Tür D, Beltagy I, Bethard S, Cotterell R, Chakraborty T, Zhou Y (eds) Proceedings of the 2021 conference of the North American chapter of the association for computational linguistics: human language technologies, NAACL-HLT 2021, Online, June 6-11, 2021, Association for Computational Linguistics, pp 2655–2668. https://doi.org/10.18653/v1/2021.naacl-main.211

Baltes S, Dumani L, Treude C, Diehl S (2018) Sotorrent: reconstructing and analyzing the evolution of stack overflow posts. In: Zaidman A, Kamei Y, Hill E (eds) Proceedings of the 15th international conference on Mining Software Repositories, MSR 2018, Gothenburg, Sweden, May 28-29, 2018, ACM, pp 319–33. https://doi.org/10.1145/3196398.3196430

Barua A, Thomas SW, Hassan AE (2014) What are developers talking about? an analysis of topics and trends in stack overflow. Empir Softw Eng 19(3):619–654

Beltagy I, Lo K, Cohan A (2019) Scibert: a pretrained language model for scientific text. arXiv:1903.10676

Bowman SR, Angeli G, Potts C, Manning CD (2015) A large annotated corpus for learning natural language inference. In: Proceedings of the 2015 conference on empirical methods in natural language processing, Association for Computational Linguistics, Lisbon, Portugal, pp 632–642. https://doi.org/10.18653/v1/D15-1075https://aclanthology.org/D15-1075

Buratti L, Pujar S, Bornea M, McCarley S, Zheng Y, Rossiello G, Morari A, Laredo J, Thost V, Zhuang Y, et al. (2020) Exploring software naturalness through neural language models. arXiv:2006.12641

Cliff N (1993) Dominance statistics: ordinal analyses to answer ordinal questions. Psychol Bull 114(3):494

Cliff N (2014) Ordinal methods for behavioral data analysis. Psychology Press

Devlin J, Chang M, Lee K, Toutanova K (2018) BERT: pre-training of deep bidirectional transformers for language understanding. arXiv:1810.04805

Feng Z, Guo D, Tang D, Duan N, Feng X, Gong M, Shou L, Qin B, Liu T, Jiang D, Zhou M (2020) CodeBERT: a pre-trained model for programming and natural languages. In: Findings of the association for computational linguistics: EMNLP 2020, Association for Computational Linguistics, Online, pp 1536–1547. https://doi.org/10.18653/v1/2020.findings-emnlp.139https://aclanthology.org/2020.findings-emnlp.139

Gehan EA (1965) A generalized wilcoxon test for comparing arbitrarily singly-censored samples. Biometrika 52(1–2):203–224

Giorgi J, Nitski O, Wang B, Bader G (2020) Declutr: deep contrastive learning for unsupervised textual representations. arXiv:2006.03659

Guo D, Ren S, Lu S, Feng Z, Tang D, Liu S, Zhou L, Duan N, Svyatkovskiy A, Fu S, Tufano M, Deng SK, Clement CB, Drain D, Sundaresan N, Yin J, Jiang D, Zhou M (2021) Graphcodebert: pre-training code representations with data flow. In: 9th International conference on learning representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021, OpenReview.net. https://openreview.net/forum?id=jLoC4ez43PZ

He J, Zhou X, Xu B, Zhang T, Kim K, Yang Z, Thung F, Irsan IC, Lo D (2024) Representation learning for stack overflow posts: how far are we? ACM Trans Softw Eng Methodol 33(3):1–24

He J, Xu B, Yang Z, Han D, Yang C, Lo D (2022) Ptm4tag: sharpening tag recommendation of stack overflow posts with pre-trained models. In: Proceedings of the 30th IEEE/ACM international conference on program comprehension, association for computing machinery, New York, NY, USA, ICPC '22, p 1-1. https://doi.org/10.1145/3524610.3527897

Hinton GE, Vinyals O, Dean J (2015) Distilling the knowledge in a neural network. arXiv:1503.02531

Huang K, Altosaar J, Ranganath R (2020) Clinicalbert: modeling clinical notes and predicting hospital readmission. arXiv:1904.05342

Huang J, Tang D, Shou L, Gong M, Xu K, Jiang D, Zhou M, Duan N (2021) Cosqa: 20,000+ web queries for code search and question answering. ArXiv:2105.13239

Huang Z, Xu W, Yu K (2015) Bidirectional lstm-crf models for sequence tagging. arXiv:1508.01991

Husain H, Wu HH, Gazit T, Allamanis M, Brockschmidt M (2020) Codesearchnet challenge: evaluating the state of semantic code search. arXiv:1909.09436

Jiang X, Zheng Z, Lyu C, Li L, Lyu L (2021) Treebert: a tree-based pre-trained model for programming language. In: de Campos CP, Maathuis MH, Quaeghebeur E (eds) Proceedings of the thirty-seventh conference on uncertainty in artificial intelligence, UAI 2021, Virtual Event, 27-30 July 2021, AUAI Press, Proceedings of Machine Learning Research, vol 161, pp 54–63. https://proceedings.mlr.press/v161/jiang21a.html

Jin D, Jin Z, Zhou JT, Szolovits P (2020) Is bert really robust? a strong baseline for natural language attack on text classification and entailment. Proceedings of the AAAI conference on artificial intelligence vol 34 pp 8018–8025

Lan T, Mao XL, Zhao Z, Wei W, Huang H (2020a) Self-attention comparison module for boosting performance on retrieval-based open-domain dialog systems. arXiv:2012.11357

Lan Z, Chen M, Goodman S, Gimpel K, Sharma P, Soricut R (2020b) Albert: A lite bert for self-supervised learning of language representations. arXiv:1909.11942

Lee J, Yoon W, Kim S, Kim D, Kim S, So CH, Kang J (2019) Biobert: a pre-trained biomedical language representation model for biomedical text mining. Bioinformati. https://doi.org/10.1093/bioinformatics/btz682

Lewis M, Liu Y, Goyal N, Ghazvininejad M, Mohamed A, Levy O, Stoyanov V, Zettlemoyer L (2020) BART: denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In: Jurafsky D, Chai J, Schluter N, Tetreault JR (eds) Proceedings of the 58th annual meeting of the association for computational linguistics, ACL 2020, Online, July 5-10, 2020, Association for Computational Linguistics, pp 7871–7880. https://doi.org/10.18653/v1/2020.acl-main.703

Li C, Xu L, Yan M, Lei Y (2020) Tagdc: a tag recommendation method for software information sites with a combination of deep learning and collaborative filtering. J Syst Softw 170:110783. https://doi.org/10.1016/j.jss.2020.110783

Lin J, Liu Y, Zeng Q, Jiang M, Cleland-Huang J (2021) Traceability transformed: generating more accurate links with pre-trained bert models. In: 2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE), IEEE, pp 324–335

Liu X, He P, Chen W, Gao J (2019a) Multi-task deep neural networks for natural language understanding. In: Korhonen A, Traum DR, Màrquez L (eds) Proceedings of the 57th conference of the association for computational linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, vol 1: Long Papers, Association for Computational Linguistics, pp 4487–4496. https://doi.org/10.18653/v1/p19-1441

Liu Y, Ott M, Goyal N, Du J, Joshi M, Chen D, Levy O, Lewis M, Zettlemoyer L, Stoyanov V (2019b) Roberta: a robustly optimized bert pretraining approach. arXiv:1907.11692

Maity SK, Panigrahi A, Ghosh S, Banerjee A, Goyal P, Mukherjee A (2019) Deeptagrec: a content-cum-user based tag recommendation framework for stack overflow. In: Azzopardi L, Stein B, Fuhr N, Mayr P, Hauff C, Hiemstra D (eds) Advances in information retrieval - 41st European conference on IR research, ECIR 2019, Cologne, Germany, April 14-18, 2019, Proceedings, Part II, Springer, Lecture Notes in Computer Science, vol 11438, pp 125–131. https://doi.org/10.1007/978-3-030-15719-7_16

Mashhadi E, Hemmati H (2021) Applying codebert for automated program repair of java simple bugs. In: 2021 IEEE/ACM 18th international conference on Mining Software Repositories (MSR), pp 505–50.https://doi.org/10.1109/MSR52588.2021.00063

Ni J, Ábrego GH, Constant N, Ma J, Hall KB, Cer D, Yang Y (2022) Sentence-t5: scalable sentence encoders from pre-trained text-to-text models. In: Muresan S, Nakov P, Villavicencio A (eds) Findings of the association for computational linguistics: ACL 2022, Dublin, Ireland, May 22-27, 2022, Association for Computational Linguistics, pp 1864–1874. https://doi.org/10.18653/v1/2022.findings-acl.146

Phan L, Tran H, Le D, Nguyen H, Annibal J, Peltekian A, Ye Y (2021) CoTexT: multi-task learning with code-text transformer. In: Proceedings of the 1st workshop on natural language processing for programming (NLP4Prog 2021), Association for Computational Linguistics, Online, pp 40–47. https://doi.org/10.18653/v1/2021.nlp4prog-1.5

Qu C, Yang L, Qiu M, Croft WB, Zhang Y, Iyyer M (2019) Bert with history answer embedding for conversational question answering. In: Proceedings of the 42nd international ACM SIGIR conference on research and development in information retrieval, pp 1133–1136

Raffel C, Shazeer N, Roberts A, Lee K, Narang S, Matena M, Zhou Y, Li W, Liu PJ (2020) Exploring the limits of transfer learning with a unified text-to-text transformer. J Mach Learn Res 21:140:1–140:67. http://jmlr.org/papers/v21/20-074.html

Reimers N, Gurevych I (2019) Sentence-bert: sentence embeddings using siamese bert-networks. arXiv:1908.10084

Rothe S, Narayan S, Severyn A (2020) Leveraging pre-trained checkpoints for sequence generation tasks. Trans Assoc Comput Linguist 8:264–28. https://doi.org/10.1162/tacl_a_00313https://aclanthology.org/2020.tacl-1.18

Sarfraz F, Arani E, Zonooz B (2021) Knowledge distillation beyond model compression. In: 2020 25th International Conference on Pattern Recognition (ICPR), IEEE, pp 6136–6143

Schmidhuber J (2015) Deep learning in neural networks: an overview. Neural Netw 61:85–117

Shi J, Yang Z, He J, Xu B, Lo D (2022) Can identifier splitting improve open-vocabulary language model of code? In: 2022 IEEE international conference on software analysis, evolution and reengineering (SANER), IEEE

Shi J, Yang Z, Kang HJ, Xu B, He J, Lo D (2024) Greening large language models of code. In: Proceedings of the 46th international conference on software engineering: software engineering in society, pp 142–153

Shi J, Yang Z, Xu B, Kang HJ, Lo D (2023) Compressing pre-trained models of code into 3 mb. In: Proceedings of the 37th IEEE/ACM international conference on automated software engineering, Association for Computing Machinery, New York, NY, USA, ASE. https://doi.org/10.1145/3551349.3556964

Sun C, Qiu X, Xu Y, Huang X (2020) How to fine-tune bert for text classification? arXiv:1905.05583

Svyatkovskiy A, Deng SK, Fu S, Sundaresan N (2020) Intellicode compose: code generation using transformer. arXiv:2005.08025

Tabassum J, Maddela M, Xu W, Ritter A (2020) Code and named entity recognition in stackoverflow. arXiv:2005.01634

Vaswani A, Shazeer N, Parmar N, Uszkoreit J, Jones L, Gomez AN, Kaiser Ł, Polosukhin I (2017) Attention is all you need. In: Advances in neural information processing systems, pp 5998–6008

Von der Mosel J, Trautsch A, Herbold S (2022) On the validity of pre-trained transformers for natural language processing in the software engineering domain. IEEE Trans Softw Eng 1–1. https://doi.org/10.1109/TSE.2022.3178469

Wang XY, Xia X, Lo D (2015) Tagcombine: recommending tags to contents in software information sites. J Comput Sci Technol 30(5):1017–1035

Wang S, Lo D, Vasilescu B, Serebrenik A (2014) Entagrec: an enhanced tag recommendation system for software information sites. In: 2014 IEEE international conference on software maintenance and evolution, pp 291–300. https://doi.org/10.1109/ICSME.2014.51

Wang S, Lo D, Vasilescu B, Serebrenik A (2018) Entagrec ++: an enhanced tag recommendation system for software information sites. Empir Softw Eng 23

Wang Y, Wang W, Joty SR, Hoi SCH (2021) Codet5: identifier-aware unified pre-trained encoder-decoder models for code understanding and generation. In: Moens M, Huang X, Specia L, Yih SW (eds) Proceedings of the 2021 conference on Empirical Methods in Natural Language Processing, EMNLP 2021, Virtual Event / Punta Cana, Dominican Republic, 7-11 November, 2021, Association for Computational Linguistics, pp 8696–8708. https://doi.org/10.18653/v1/2021.emnlp-main.685

Wang S, Xu Y, Fang Y, Liu Y, Sun S, Xu R, Zhu C, Zeng M (2022a) Training data is more valuable than you think: a simple and effective method by retrieving from training data. In: Proceedings of the 60th annual meeting of the association for computational linguistics (Volume 1: Long Papers), pp 3170–3179

Wang X, Zhou K, rong Wen J, Zhao WX (2022b) Towards unified conversational recommender systems via knowledge-enhanced prompt learning. Proceedings of the 28th ACM SIGKDD conference on knowledge discovery and data mining

Williams A, Nangia N, Bowman S (2018) A broad-coverage challenge corpus for sentence understanding through inference. In: Proceedings of the 2018 conference of the north american chapter of the association for computational linguistics: human language technologies, vol 1 (Long Papers), Association for Computational Linguistics, New Orleans, Louisiana, pp 1112–112. https://doi.org/10.18653/v1/N18-1101https://aclanthology.org/N18-1101

Xia X, Lo D, Wang X, Zhou B (2013) Tag recommendation in software information sites. In: Proceedings of the 10th working conference on mining software repositories, IEEE Press, MSR '13, pp 287-296

Xu B, Hoang T, Sharma A, Yang C, Xia X, Lo D (2021) Post2vec: learning distributed representations of stack overflow posts. IEEE Trans Softw Eng 1. https://doi.org/10.1109/TSE.2021.3093761

Yang C, Xu B, Khan Younus J, Uddin G, Han D, Yang Z, Lo D (2022) Aspect-based api review classification: how far can pre-trained transformer model go? In: 29th IEEE international conference on software analysis, evolution and reengineering(SANER), IEEE

Zhang T, Xu B, Thung F, Haryono SA, Lo D, Jiang L (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, pp 70–80

Zhou P, Liu J, Yang Z, Zhou G (2017) Scalable tag recommendation for software information sites. 2017 IEEE 24th International Conference on Software Analysis. Evolution and Reengineering (SANER), IEEE, pp 272–282

Zhou P, Liu J, Liu X, Yang Z, Grundy J (2019) Is deep learning better than traditional approaches in tag recommendation for software information sites? Inf Softw Technol 109:1–13. https://doi.org/10.1016/j.infsof.2019.01.002

Zhou X, Han D, Lo D (2021) Assessing generalizability of codebert. In: 2021 IEEE International Conference on Software Maintenance and Evolution (ICSME), IEEE, pp 425–436

Zhou X, Kim K, Xu B, Han D, He J, Lo D (2023a) Generation-based code review automation: how far are we'. In: 2023 IEEE/ACM 31st International Conference on Program Comprehension (ICPC), IEEE, pp 215–226

Zhou X, Xu B, Han D, Yang Z, He J, Lo D (2023b) Ccbert: Self-supervised code change representation learning. In: 2023 IEEE International Conference on Software Maintenance and Evolution (ICSME), IEEE, pp 182–193

Zhu Y, Kiros R, Zemel R, Salakhutdinov R, Urtasun R, Torralba A, Fidler S (2015) Aligning books and movies: towards story-like visual explanations by watching movies and reading books. In: The IEEE International Conference on Computer Vision (ICCV)

Junda He

Bowen Xu

Zhou Yang

DongGyun Han

Chengran Yang

Jiakun Liu

Zhipeng Zhao

David Lo

Authors and Affiliations

Junda He[1] · Bowen Xu[2] · Zhou Yang[1] · DongGyun Han[3] · Chengran Yang[1] · Jiakun Liu[1] · Zhipeng Zhao[4] · David Lo[1]

✉ Jiakun Liu
jkliu@smu.edu.sg

Junda He
jundahe@smu.edu.sg

Bowen Xu
bxu22@ncsu.edu

Zhou Yang
zyang@smu.edu.sg

DongGyun Han
DongGyun.Han@rhul.ac.uk

Chengran Yang
cryang@smu.edu.sg

Zhipeng Zhao
zhipeng.zhao@di.ku.dk

David Lo
davidlo@smu.edu.sg

1 School of Computing and Information Systems, Singapore Management University, Singapore, Singapore

2 North Carolina State University, Raleigh, NC, USA

3 Department of Computer Science, Royal Holloway, University of London Egham, Egham, UK

4 Department of Computer Science, University of Copenhagen, Copenhagen, Denmark