Representation Learning for Stack Overflow Posts: How Far are We?

JUNDA HE, Singapore Management University, Singapore
XIN ZHOU, Singapore Management University, Singapore
BOWEN XU*, North Carolina State University, USA
TING ZHANG, Singapore Management University, Singapore
KISUB KIM*, Singapore Management University, Singapore
ZHOU YANG, Singapore Management University, Singapore
FERDIAN THUNG, Singapore Management University, Singapore
IVANA CLAIRINE IRSAN, Singapore Management University, Singapore
DAVID LO, Singapore Management University, Singapore

The tremendous success of Stack Overflow has accumulated an extensive corpus of software engineering knowledge, thus motivating researchers to propose various solutions for analyzing its content. The performance of such solutions hinges significantly on the selection of representation models for Stack Overflow posts. As the volume of literature on Stack Overflow continues to burgeon, it highlights the need for a powerful Stack Overflow post representation model and drives researchers' interest in developing specialized representation models that can adeptly capture the intricacies of Stack Overflow posts. The state-of-the-art (SOTA) Stack Overflow post representation models are Post2Vec and BERTOverflow, which are built upon neural networks such as convolutional neural network (CNN) and transformer architecture (e.g., BERT). Despite their promising results, these representation methods have not been evaluated in the same experimental setting. To fill the research gap, we first empirically compare the performance of the representation models designed specifically for Stack Overflow posts (Post2Vec and BERTOverflow) in a wide range of related tasks, i.e., tag recommendation, relatedness prediction, and API recommendation. The results show that Post2Vec cannot further improve the state-of-the-art techniques of the considered downstream tasks, and BERTOverflow shows surprisingly poor performance. To find more suitable representation models for the posts, we further explore a diverse set of transformer-based models, including (1) general domain language models (RoBERTa, Longformer, GPT2) and (2) language models built with software engineering-related textual artifacts (CodeBERT, GraphCodeBERT, seBERT, CodeT5, PLBart, and CodeGen). This exploration shows that models like CodeBERT and RoBERTa are suitable for representing Stack Overflow posts. However, it also illustrates the "No Silver Bullet" concept, as none of the models consistently wins against all the others. Inspired by the findings, we propose SOBERT, which employs a simple yet effective strategy to improve the representation models of Stack Overflow posts by continuing the pre-training phase with the

*Corresponding Authors

Authors' addresses: Junda He, jundahe@smu.edu.sg, Singapore Management University, 80 Stamford Rd., 178902, Singapore, Singapore; Xin Zhou, xinzhou.2020@phdcs.smu.edu.sg, Singapore Management University, 80 Stamford Rd., 178902, Singapore, Singapore; Bowen Xu, bxu22@ncsu.edu, North Carolina State University, Raleigh, USA; Ting Zhang, tingzhang.2019@phdcs.smu.edu.sg, Singapore Management University, 80 Stamford Rd., 178902, Singapore, Singapore; Kisub Kim, kisubkim@smu.edu.sg, Singapore Management University, 80 Stamford Rd., 178902, Singapore, Singapore; Zhou Yang, zyang@smu.edu.sg, Singapore Management University, 80 Stamford Rd., 178902, Singapore, Singapore; Ferdian Thung, ferdianthung@smu.edu.sg, Singapore Management University, 80 Stamford Rd., 178902, Singapore, Singapore; Ivana Clairine Irsan, ivanairsan@smu.edu.sg, Singapore Management University, 80 Stamford Rd., 178902, Singapore, Singapore; David Lo, davidlo@smu.edu.sg, Singapore Management University, 80 Stamford Rd., 178902, Singapore, Singapore.

ACM 1049-331X/2023/12-ART
https://doi.org/10.1145/3635711

textual artifact from Stack Overflow. The overall experimental results demonstrate that SOBERT can consistently outperform the considered models and increase the state-of-the-art performance significantly for all the downstream tasks.

CCS Concepts: • **Computing methodologies** → *Knowledge representation and reasoning*; • **Software and its engineering** → **Software development process management**.

Additional Key Words and Phrases: Stack Overflow, Transformers, Pre-trained Models

1 INTRODUCTION

Serving as the most popular software question and answer (SQA) forum, Stack Overflow (SO) has dramatically influenced modern software development practice. As of August 2023, the forum has accumulated more than 23 million questions and 35 million answers[1]. Stack Overflow is broadly recognized as an invaluable knowledge base and supplemental resource for the software engineering (SE) domain [6, 18, 52, 55], which triggered the increased interest of researchers and software developers in a wide range of Stack Overflow post-related tasks, for example, recommendation of post tags (aka. tag recommendation) [18], recommendation of APIs according to a natural language query (aka. API recommendation) [6], and the identification of related posts (aka. relatedness prediction) [55].

An essential step in yielding promising results for these Stack Overflow-related tasks is to obtain suitable representations of the posts. A beneficial Stack Overflow representation model can capture the semantic concept of the posts and reveal more explanatory features from the hidden dimensions. As the volume of SE literature on solving Stack Overflow-related tasks [18, 52, 55] continues to burgeon, it has underscored the demand for a quality Stack Overflow representation.

Over the years, numerous representation models have been specifically proposed for modeling Stack Overflow posts. Xu et al. proposed Post2Vec [54], a CNN-based [24] representation model that leverages the tags of a post to guide the learning process and models the post as the combination of three complementary components (i.e., title, description, and code snippet). Their experimental results demonstrate that it can substantially boost the performance for a wide range of Stack Overflow posts-related tasks [1, 3, 6]. Tabassum et al. [44] leveraged the more advanced transformer architecture and pre-trained BERTOverflow based on 152 million sentences from Stack Overflow. The results demonstrate that the embeddings generated by BERTOverflow have led to a significant improvement over other off-the-shelf models (e.g., ELMo [34] and BERT [13]) in the software named entity recognition (NER) task.

Although these existing Stack Overflow-specific methods have been proven to be beneficial, the effectiveness of Post2Vec is only evaluated on limited solutions (i.e., Support Vector Machine [55] and Random Forest [5]) and BERTOverflow only experimented for the NER task. These motivate us to further study the performance of existing Stack Overflow-specific representation models on a diverse set of tasks. Unexpectedly, we found that both Post2Vec and BERTOverflow perform poorly. Such findings motivate us to explore the effectiveness of a larger array of representation techniques in modeling Stack Overflow posts.

In addition to the aforementioned Stack Overflow-specific representation models, we further consider nine transformer-based language models that could be potentially suitable for post representation learning. These models can be classified into two types: SE domain-specific models and general domain models. SE domain-specific models are trained with SE-related contents (i.e., Github repositories) and are suitable for capturing the semantics of technical jargon of the SE domain. We consider six SE domain-specific models: CodeBERT [14], GraphCodeBERT [16], seBERT [49], CodeGen [32], CodeT5 [50], and PLBart [2]. We also include models from the general domain as they are usually trained with a more diverse amount of data than domain-specific models. For general domain models, we consider RoBERTa [28], Longformer [4], and GPT2 [38].

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

We evaluate the performance of the aforementioned representation models on multiple Stack Overflow-related downstream tasks (i.e., tag recommendation [54], API recommendation [6], and relatedness prediction [55]). Furthermore, we build SOBERT, a stronger transformer-based language model for modeling Stack Overflow posts. Our experimental results reveal several interesting findings:

(1) *Existing Stack Overflow post representation techniques fail to improve the SOTA performance of considered tasks.* Xu et al. demonstrated that the addition of the feature vectors generated by Post2Vec is beneficial for improving the post representation for traditional machine learning techniques. However, we discover that appending the feature vectors from Post2Vec [54] does not derive a beneficial effect on considered deep neural networks. Furthermore, we reveal that the embedding generated by BERTOverflow could only achieve reasonable performance in the API recommendation task and give surprisingly poor performance in the tag recommendation task.
(2) *Among all the considered models, none of them can always perform the best.* According to our experiment results, although several representation models can outperform the SOTA approaches, none can always perform the best. As a result, this motivates us to propose a new model for representing Stack Overflow posts.
(3) *Continued pre-training based on Stack Overflow textual artifact develops a consistently better representation model.* We propose SOBERT by further pre-training with Stack Overflow data. The overall results show that SOBERT consistently boosts the performance in all three considered tasks, implying a better representation.

Overall, we summarize the contributions of our empirical study as follows:

(1) We comprehensively evaluate the effectiveness of eleven representation models for Stack Overflow posts in three downstream tasks.
(2) We propose SOBERT by pre-training based on posts from Stack Overflow and show that SOBERT consistently outperforms other representation models in multiple downstream tasks.
(3) We derive several insightful lessons from the experimental results to the software engineering community.

The rest of the paper is organized as follows. Section 2 categorizes representation learning models into three groups and briefly describes them. We formulate the downstream tasks (i.e., tag recommendation, API recommendation, relatedness prediction) and their corresponding state-of-the-art method in Section 3. Section 4 introduces our research questions and the experiment settings. In Section 5, we answer the research questions and report the experiment results. Section 6 further analyzes the result and elaborates the insights with evidence. Section 7 describes related works, and Section 8 summarizes this study.

2 REPRESENTATION LEARNING MODELS

In this section, we summarize the considered representation models in this paper. We explore a wide range of techniques across the spectrum of representing Stack Overflow posts, including two Stack Overflow-specific post representation models (Post2Vec [54] and BERTOverflow [44]), six SE domain-specific transformer-based pre-trained representation models (PTM) (CodeBERT [14], GraphCodeBERT [16], seBERT [49], CodeT5 [50], PLBart [2], and CodeGen [32]) and three transformer-based PTMs from the general domain (RoBERTa [28] Longformer [4], and GPT2 [38]).

2.1 Transformer-based Language Models

Transformer-based language models have revolutionized the landscape of representation learning in natural language processing (NLP) [13, 28, 38]. Their efficacy in capturing text semantics has led to unparalleled performance in various applications, such as sentiment analysis [43], POS tagging [45], and question answering [36]. The vanilla transformer architecture [48] is composed of the encoder and decoder components. Based on the usage of these components, transformer-based language models can be categorized into three types: encoder-only, decoder-only, and encoder-decoder models.

Encoder-only models exclusively leverage the encoder stacks of the vanilla transformer [48] architecture. BERT [13] stands as a prominent encoder-only representation model, which learns a bidirectional contextual representation of text. BERT proposes the *Masked Language Modeling* (MLM) task at the pre-training phase. In MLM, the input data is corrupted by randomly masking 15% of the tokens, and then the BERT model learns to reconstruct the original data by predicting the masked words. BERT is extensively pre-trained on large-scale datasets, which learn a meaningful representation that is reusable for various tasks, thus eliminating the process of training language models from scratch and saving time and resources.

In contrast, *Decoder-only models* consist solely of the decoder components of the original transformer architecture. A notable instance of such models is the GPT [37], GPT operates under a causal language modeling (CLM) framework during its training phase. CLM is a strategy where the model predicts the next token in a sequence while only considering preceding tokens. In other words, this design restricts the model from accessing future tokens in the sequence.

Bridging the above approaches, textitEncoder-decoder models integrate both the encoder and decoder components of the transformer architecture. Popular encoder-decoder models involve T5 [39] and BART [25]. The T5 model [39] advocates a unified text-to-text framework that converts various language tasks into a consistent text-to-text format. T5 is pre-trained on the Colossal Clean Crawled Corpus [39], along with a mixture of unsupervised and supervised pre-training tasks. On the other hand, BART [25] introduces a variety of noising functions to corrupt the initial input sequence (i.e., token deletion, document rotation, and sentence shuffling) during the pre-training phase. By corrupting the original sequence through these mechanisms, BART is trained to restore the original input.

2.2 Existing Representation Models for Stack Overflow Posts

Post2Vec [54] is the latest approach proposed specifically for Stack Overflow post representation learning [54]. Post2Vec is designed with a *triplet* architecture to process three components of a Stack Overflow post (i.e., title, text, and code snippets). It leverages Convolutional Neural Networks (CNNs) as feature extractors to encode the three components separately. The corresponding three output feature vectors are then fed to a feature fusion layer to represent the post. In the end, Post2Vec uses tag information of the post, which is considered as the post's general semantic meaning to supervise the representation learning process. Xu et al. demonstrated that the representation learned by Post2Vec can enhance the feature vectors for Stack Overflow-related downstream tasks (e.g., relatedness prediction and API recommendation). For each downstream task, the vector representation learned by Post2Vec is combined with the feature vector produced by the corresponding state-of-the-art approach to form a new feature vector. The new feature vector is used to boost the performance of the corresponding model for the task. Following the experiment settings of Xu et al., we use Post2Vec as a complementary feature vector to the state-of-the-art approach in this paper. Specifically, we concatenate the post representation generated by Post2Vec to the original feature vector of the state-of-the-art approach. This combined feature vector is employed in further training.
BERTOverflow [44] keeps the original BERT architecture, and it leverages 152 million sentences and 2.3 billion tokens from Stack Overflow to pre-train Stack Overflow-specific word embeddings. The authors have leveraged the embedding generated by BERTOverflow to implement a software-related named entity recognizer (SoftNER). The performance of SoftNER is experimented with the name entity recognition (NER) task for the software engineering domain, focusing on identifying code tokens or programming-related named entities that appear within SQA sites like Stack Overflow. The results show that BERTOverflow outperforms all other models in the proposed task.

2.3 Representation Models from SE domain

CodeBERT [14] is a SE knowledge-enriched bi-modal pre-trained model, which is capable of modeling both natural languages (NL) and programming languages (PL).The CodeBERT model has shown great effectiveness in a diverse range of SE domain-specific activities, for example, code search [14], traceability prediction [27], and code translation [8]. CodeBERT inherits the architecture of BERT [13], and it continues pre-training based on the checkpoint of RoBERTa [28] with the NL-PL data pairs obtained from the CodeSearchNet dataset [22]. It has two popular pre-training objectives: Masked Language Modeling (MLM) and Replaced Token Detection (RTD) [10]. Rather than masking the input like MLM, RTD corrupts the input by replacing certain tokens with plausible substitutes. RTD then predicts if each token in the altered input was replaced or remained unchanged. The eventual loss function for CodeBERT at the pre-training stage is the combination of both MLM and RTD objectives, where θ denotes the model parameters:

$$\underset{\theta}{\text{mini}}(\mathcal{L}_{RTD}(\theta) + \mathcal{L}_{MLM}(\theta)) \tag{1}$$

GraphCodeBERT [16] incorporates a hybrid representation in source code modeling. Apart from addressing the pre-training process over NL and PL, GraphCodeBERT utilizes the data flow graph of source code as additional inputs and proposes two structure-aware pre-training tasks (i.e., Edge Prediction and Node Alignment) aside from the MLM prediction task. GraphCodeBERT is evaluated in code search [14], clone detection [51], code translation [8], and code refinement [46], respectively. It outperforms CodeBERT and all the other baselines, including RoBERTa (code version) [16], Transformer [48], and LSTM [19].
seBERT [49] aims to advance the previous PTMs in the SE context with a larger model architecture and more diverse pre-training data. The authors pre-trained seBERT using the BERT$_{LARGE}$ architecture, i.e., with 24 layers, a hidden layer size of 1024, and 16 self-attention heads, with a total of 340 million parameters. seBERT is pre-trained with more than 119GB of data from four data sources, i.e., Stack Overflow posts, GitHub issues, Jira issues, and Github commit messages. The model's effectiveness is verified in three classification tasks, i.e., issue type prediction, commit intent prediction, and sentiment mining. The authors observe the experiment results showing that seBERT is significantly better than BERToverflow in these tasks.
CodeGen [32] is a pre-eminent decoder-only transformer-based for program synthesis, it undergoes pre-training on an extensive dataset comprising both natural language and programming languages. This model innovates a multi-turn programming synthesis paradigm and creates a comprehensive benchmark for multi-turn programming tasks. The multi-turn program synthesis approach involves users and the model together in multiple steps. The user communicates with the model by progressively providing specifications in natural language while receiving responses from the model in the form of synthesized subprograms. CodeGen demonstrates state-of-the-art performance on Python code generation on HumanEval [7] and a set of tasks in the multi-turn programming benchmark. The model is pre-trained on BigQuery dataset[2]. This dataset contains GitHub repositories with multiple programming languages, including C, C++, Go, Java, JavaScript, and Python.
CodeT5 [50] is an encoder-decoder PTM that is designed to better consider the code semantics conveyed from the identifiers from code. It has the same architecture as T5 [39] and employs a multi-task training process to support both code understanding and generation tasks. CodeT5 utilizes a novel identifier-aware pre-training task that enables the model to distinguish which code tokens are identifiers and to recover them when they are masked. To improve the NL-PL alignment, CodeT5 further incorporates a bimodal dual learning objective for a bidirectional conversion between natural languages and programming languages.
PLBart [2] is also an encoder-decoder PTM, which is capable of performing a broad spectrum of program-related understanding and generation tasks. PLBART employs the same architecture as BART [25] and is pre-trained on a large collection of Java and Python functions and associated natural language documentation from Github

[2]https://cloud.google.com/bigquery/public-data

repositories and Stack Overflow posts. Experiments showed that PLBART can achieve promising performance in code summarization, code generation, and code translation.

2.4 Representation Models from General Domain

RoBERTa [28] is a replication study on the pre-training objectives of BERT [13], and analyzed the impact of key hyper-parameters. The insights from the replication study have led to the development of RoBERTa, which is an improved version of BERT. In comparison with BERT, RoBERTa has made several modifications to the pre-training stage: (1) training with larger batch size, more data, and longer training time; (2) abandoning the next sentence prediction (NSP) task of BERT and showed that removal of NSP slightly improves the model efficiency; (3) training with longer sequences; (4) masking the training data dynamically rather than statically.
Longformer [4] aims to alleviate the limitation of transformer-based models in processing long sequences. The self-attention mechanism of the transformer suffers from the $O(n^2)$ quadratic computational complexity problem, which restricts the ability of transformer-based models to model long sequences. Pre-trained models like BERT [13] and RoBERTa [28] only accept a maximum input of 512 tokens. Longformer leverages a combination of sliding window attention and global attention mechanism such that the computational memory consumption scales linearly as the sequence becomes longer. In contrast to models like RoBERTa and CodeBERT, which could only accept a maximum of 512 tokens as input, Longformer supports sequences of length up to 4,096. Similar to CNN [24], Longformer lets each input token only attend to surrounding neighbors that are within a fixed window size. Denoting the window size as w, each token could only attend to $\frac{1}{2}w$ tokens on both sides, thus decreasing the computation complexity to $O(n \times w)$. However, the sliding window may compromise the performance as it cannot capture the whole context. To compensate for the side-effect, global tokens are selected. Such tokens are implemented with global attention, which attends to all other tokens, and other tokens also attend to the global tokens.
GPT2 [38] is a decoder-only model. It is trained with a simple objective: predicting the next word, given all of the previous words within the text. GPT-2 is trained on a dataset of 8 million web pages. The diversity of the dataset causes this simple goal to contain naturally occurring demonstrations of many tasks across diverse domains. GPT-2 is a direct scale-up of GPT, with more parameters and trained on more than 10 times the amount of data.

3 DOWNSTREAM TASKS

In this section, we formulate the target problems that are used to measure the effectiveness of the representation models and then describe the corresponding state-of-the-art solution. We select multiple Stack Overflow-related downstream tasks, which have been popular research topics for Stack Overflow posts. To be more specific, we consider: *Tag Recommendation* [18, 54], *API Recommendation* [6, 52] and *Relatedness Prediction* [33, 55], covering a multi-label classification problem, a ranking problem, and multi-class classification problem. All selected tasks operate on the abstraction of a post, which could be benefited from a high-quality Stack Overflow post representation.

3.1 Tag Recommendation

The user-annotated tags of a Stack Overflow post serve as helpful metadata and have a critical role in organizing the contents of Stack Overflow posts across different topics. Suitable tags precisely summarize the message of a post, while redundant tags and synonym tags make it more difficult in maintaining the content of the site. A tag recommendation system could effectively simplify the tagging process and minimize the effect of manual errors, therefore, avoiding problems like tag synonyms and tag redundancy.

3.1.1 Task Formulation. We formulate the tag recommendation task as a *multi-label classification problem.* Given $\mathcal{X}$ as the corpus of Stack Overflow posts, and $\mathcal{Y}$ denotes the total collection of tags, we represent each post as x_i, where $0 \leq i \leq |X|, i \in \mathbb{N}$ and the tags of each post as $y_i \subset \mathcal{Y}$. The goal is to recommend the most relevant set of tags y_i to x_i.

3.1.2 State-of-the-art technique. PTM4Tag [18] leverages three pre-trained models to solve the tag recommendation problem. The three pre-trained models are responsible for modeling the title, description, and code snippet, independently.

3.2 API Recommendation

Modern software development process heavily relies on third-party APIs, which leads to the research of an automated API recommendation approach that is intended to simplify the process of API search [53]. Questions related to APIs are one of the most viewed topics on Stack Overflow [20]. Stack Overflow consists of an enormous amount of discussion about API usage. Developers are more intended to search for relevant Stack Overflow posts and pick out the APIs that seem useful in the discussions [21] rather than checking API documentation. Thus, it makes Stack Overflow the primary source for building a dataset of the API recommendation task.

3.2.1 Task Formulation. We follow the exact task definition as the previous literature [15, 20, 52]. Given a natural language (NL) query that describes programming requirements, the goal is to recommend relevant APIs that implement the function for the query. Thus, the task aims to inform developers which API to use for a programming task. Formally speaking, given the corpus of natural language query Q, we denote each query as q_i. The goal of the API recommendation system is to find a set of relevant APIs $y_i \subset \mathcal{Y}$ for q_i, where $\mathcal{Y}$ is the total set of available APIs.

3.2.2 State-of-the-art technique. Wei et al. [52] proposed CLEAR, an automated approach that recommends API by embedding queries and Stack Overflow posts with a PTM (distilled version of the RoBERTa[3]). Given a NL query, CLEAR firstly picks a sub-set of candidate Stack Overflow posts based on the embedding similarity to reduce the search space. Then, CLEAR ranks the candidate Stack Overflow posts and recommends the APIs from the top-ranked Stack Overflow posts.

3.3 Relatedness Prediction

The notion of a knowledge unit (KU) is defined as a set containing a question along with all its answers [3, 33, 55]. To find a comprehensive technical solution for a given problem, developers usually need to summarize the information from multiple related KUs. However, searching for related KUs can be time-consuming as the same question can be rephrased in many different ways. Thus, researchers have proposed several techniques to automate the process of identifying the related KUs [3, 33, 55], which could significantly improve the efficiency of the software development cycle.

3.3.1 Task Formulation. The task is commonly formulated as a multi-class classification problem [3, 33, 55]. The relatedness between questions is classified into four classes, from the most relevant to irrelevant, which are:

- *Duplicate*: Two KUs correspond to a pair of semantically equivalent questions. The answer of one KU can also be used to answer another KU.
- *Direct*: One KU is beneficial in answering the question in another KU, for example, by explaining certain concepts and giving examples.
- *Indirect*: One KU provides relevant information but does not directly answer the questions of another KU.
- *Isolated*: The two KUs are semantically uncorrelated.

Given the set K of all knowledge units, the goal of relatedness prediction is to predict the degree of relatedness between any two KUs, k_i and k_j. The relatedness class is denoted as C, where $C = \{Duplicate, Direct, Indirect, Isolated\}$.

[3]https://huggingface.co/distilroberta-base

Table 1. Examples of duplicate, direct, indirect knowledge units pairs for the relatedness prediction task.

	Post ID	Title
Original KU	513832	*How do I compare strings in Java?*
Duplicate KU	3281448	*Strings in Java : equals vs ==*
Direct KU	34509566	*"==" in case of String concatenation in Java*
Indirect KU	11989261	*Does concatenating strings in Java always lead to new strings being created in memory?*

Formally, the task of relatedness prediction is defined as obtaining the function R, such that $R(k_i, k_j) = c$, where $c \in C$.

In Table 1, we demonstrate examples on pairs of KUs with different relatedness.

- Original KU: This KU addresses the topic of string comparison in Java.
- Duplicate KU: This KU introduces the difference of "equals" and "==". Essentially, it offers a varied perspective on the same topic of string comparison in Java.
- Direct KU: This KU focuses on the behavior of "==" during string concatenation in Java. As it provides insights directly beneficial to understanding the original KU's topic without being a duplicate, it is categorized as directly related.
- Indirect KU: This KU discusses the memory allocation during string concatenation in Java. While it doesn't directly address the main topic of string comparison, its relevance to the direct KU concerning string operations classifies it as an indirectly related unit.

3.3.2 State-of-the-art technique. Recently, Pei et al. introduced ASIM [33], which yielded state-of-the-art performance in the relatedness prediction task. Pei et al. pre-trained word embeddings specialized to model Stack Overflow posts with a corpus collected from the Stack Overflow data dump. Then ASIM uses BiLSTM [42] to extract features from Stack Overflow posts and implements the attention mechanism to capture the semantic interaction among the KUs.

4 RESEARCH QUESTIONS AND EXPERIMENTAL SETTINGS

In this section, we first introduce our research questions and then describe the corresponding experiment settings.

4.1 Research Questions

RQ1. How effective are the existing Stack Overflow post representation models? Various methods have been proposed in modeling Stack Overflow posts. However, there is still a lack of analysis of the existing Stack Overflow-specific representation methods. For instance, Xu et al. [54] have demonstrated that Post2Vec is effective in boosting the performance of traditional machine learning algorithms, i.e., support vector machine (SVM) and Random Forest. However, the efficacy of Post2Vec in facilitating deep learning-based models has not yet been investigated. Moreover, Tabassum et al. [44] only leveraged the embeddings from BERTOverflow in the software-related NER task, but not for other popular Stack Overflow-related tasks. In light of this research gap, we aim to evaluate the current Stack Overflow-specific representation methods for popular Stack Overflow-related tasks under the same setting for this research question.

RQ2. How effective are the popular transformer-based language models for the targeted downstream tasks? In addition to the existing Stack Overflow representation models, we explore the effectiveness of a wider spectrum of representation models. transformer-based language models have shown great performance and generalizability in representation learning. Representations generated by such models have demonstrated promising performance in a broad range of tasks with datasets of varying sizes and origins. Borrowing the best-performing representation models from various domains and investigating their performance can derive interesting results, as recent literature [57, 58] have revealed that they are potentially great candidates for representing posts as well. This motivates us to employ RoBERTa [28] and Longformer [4] from the general domain and CodeBERT [14], GraphCodeBERT [16], and seBERT [49] from the SE domain. We set up the exact same experimental settings for each model.

RQ3. Is further pre-training on Stack Overflow data helpful in building a better model? Further pre-trained models with domain-specific corpus have been common practice in the NLP domain, however, their effectiveness is not verified for representing Stack Overflow posts. In this RQ, we introduce SOBERT, which is obtained by continuing the pre-training process on CodeBERT with Stack Overflow data, and we aim to investigate whether further pre-training with Stack Overflow data improves the performance.

4.2 Experimental Settings

4.2.1 ***Tag Recommendation.***

Dataset.
The dataset used by He et al. [18] in the training of PTM4Tag only includes the Stack Overflow posts dated before September 5, 2018. To address this limitation, we use the Stack Overflow data dump released in August of 2022 to construct a new dataset for our experiment. Ideally, a tag recommendation approach should only learn from high-quality questions. Therefore, we remove the low-quality questions when constructing the dataset. According to the classification criteria of question quality defined by Ponzanelli et al. [35], we first filter out the questions that do not have an accepted answer and further remove the questions with a score of less than 10. Additionally, we exclude the rare tags and rare posts. Previous literature in tag recommendation [18, 54] has defined a tag as rare if it occurs less than 50 times within the dataset, and a post is considered rare if all of its tags are rare tags. The usage of rare tags is discouraged since it implies the unawareness of the tag among developers. We follow the same definition as the previous literature and set the frequency threshold for rare tags as 50. In the end, the resultant dataset consists of 527,717 posts and 3,207 tags. We split the dataset into a training set, a validation set, and a test set according to the 8:1:1 ratio, which corresponds to 422,173, 52,772, and 52,772 posts, respectively.

During the training process, we only consider the question posts from Stack Overflow and ignore the answer posts. We check the post-IDs of the question posts in our dataset to ensure that each post has a unique post-ID. Each question post consists of two components, which are the title and body. The code snippets within the body of a post are enclosed in HTML tags `<pre><code>` and `</code></pre>`, we cleaned the redundant HTML tags with regular expression, we first clean the HTML tag by using regular expressions `<pre><code>([\s\S]*?)<//code><//pre>`. After that, we concatenate the title and body together to form the final input data. Thus, our input data has integrated the title, natural languages in the body, and code snippets in the body. This input data is then fed into the tokenizer of the representation model to be tokenized.

Evaluation Metrics.
We report the performance for this task using Precision@k, Recall@k, and F1-score@k, where k indicates the top-k recommendations. Such metrics are extensively used in previous works [18, 26, 54, 59], and we calculate the average score for each of them. Mathematically speaking, the evaluation metrics are computed as follows:

$$Precision@k = \frac{|\mathrm{Tag}_{\mathrm{True}} \cap \mathrm{Tag}_{\mathrm{Predict}}|}{k}$$

$$Recall@k_i = \begin{cases} \frac{|\text{Tag}_{\text{True}} \cap \text{Tag}_{\text{Predict}}|}{k} & \text{if } |\text{Tag}_{\text{True}}| > k \\ \frac{|\text{Tag}_{\text{True}} \cap \text{Tag}_{\text{Predict}}|}{|\text{Tag}_{\text{True}}|} & \text{if } |\text{Tag}_{\text{True}}| \leq k \end{cases}$$

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

In the above formulas, Tag_{True} refers to the ground truth tags and $\text{Tag}_{\text{Predict}}$ refers to the predicted tags. Notice that the above formula of Recall@k is determined by conditions since Recall@k naturally disfavors small k. The revisited Recall@k has been widely adopted in previous experiments of tag recommendation [18, 54, 59]. Since Stack Overflow posts cannot have more than 5 tags, we report the results by setting the k as 1, 3, and 5.

Implementation Details.

For Longformer, we set the maximum accepted input sequence as 1,024, and for other transformer-based language models the maximum input sequence is set as 512. This setting of the input sequence is kept the same for the other two tasks (API recommendation and relatedness prediction). For encoder-decoder and decoder-only representation models, we select the last decoder hidden state as the representation following previous literature [25, 50].

We set the learning rate as 5e-5, batch size as 512, epoch number as 30, and use the Adam optimizer to update the parameters. We save the model at the end of each epoch and select the model with the smallest validation loss to run the evaluation.

4.2.2 ***API Recommendation.***

Dataset.
We use the BIKER dataset [21] crafted by Huang et al., which is the same dataset used by Wei et al. [52]. The training set contains 33K questions with corresponding relevant APIs in the accepted answers. The test dataset contains manually labeled questions from Stack Overflow, which are looking for API to solve programming problems and labeled the ground-truth API for these questions based on their accepted answers.

When creating the Biker dataset, Huang et al. first select question posts from Stack Overflow satisfying the following three criteria: (1) the question has a positive score, (2) at least one answer to the question contains API entities (3) the answer has a positive score. Huang et al. then manually inspected the collected questions and removed the questions that were not about searching APIs for programming tasks.

Huang et al. aim to create queries that do not have too many words, thus only the titles of the Stack Overflow posts are used as queries. The ground truth APIs were extracted from the code snippets in the accepted answers in filtered posts. Again, the extracted APIs were manually checked to ensure their correctness. Eventually, the test dataset contains 413 questions along with their ground truth APIs after the manual labeling process. The titles of these questions are used as the queries for API searching.

Evaluation Metrics.
We use the same evaluation metrics as previous literature [20, 52] for the API recommendation task. The metrics are Mean Reciprocal Rank (MRR), Mean Average Precision (MAP), Precision@k and Recall@k. Mathematically, MRR is defined as:

$$MRR = \frac{1}{|Q|} \sum_{i=1}^{|Q|} \frac{1}{rank_i}$$

where Q refers to all queries and $rank_i$ refers to the rank position of the first relevant API from the recommended API list for the query$_i$. For a given query, MRR gives a score of $\frac{1}{rank}$, where $rank$ is the ranking of the first correct

API in the recommended list. In other words, the score of MRR is inversely proportional to the rank of the first correct API. MAP is the mean of average precision scores (*AveP*) for each query. While MRR gives the score based on the ranking of the first correct answer, MAP considers the ranks of all correct answers to measure the quality of the recommended list. Mathematically, MAP is defined as:

$$MAP = \frac{1}{|Q|} \sum_{i=1}^{|Q|} AveP(i)$$

$AveP(i)$ itself is defined as:

$$AveP(i) = \frac{1}{|K|} \sum_{k \in K} \frac{num(k)}{k}$$

where K is the set of ranking positions of the relevant APIs from the APIs list of query$_i$, and $num(k)$ represents the number of relevant API in the top-k.

Differently from tag recommendation, the Recall@k metrics used in this task follow the conventional definition, which is:

$$Recall@k = \frac{|\mathrm{API}_{\mathrm{True}} \cap \mathrm{API}_{\mathrm{Predict}}|}{|\mathrm{API}_{\mathrm{True}}|}$$

To be consistent with Wei et al. [52], we use k ∈ 1, 3, 5.

Implementation Details.
CLEAR shows state-of-the-art performance in the API recommendation task by leveraging BERT sentence embedding and contrastive learning. The original architecture of CLEAR is implemented based on DistilRoBERTa [4] during the training process. In this study, we also explore the effectiveness of other representation methods by replacing the embedding of DistilRoBERTa in CLEAR. For Post2Vec, we concatenate the post representation from Post2Vec to the original implementation of CLEAR.

For this task, we set the batch size as 256, and the epoch number as 30. Same to the description in Sec 4.2.1, we select the model with the smallest validation loss to run the test set.

4.2.3 ***Relatedness Prediction.***

Dataset.
The experiments are conducted based on the KUs dataset provided by Shirani et al. [3]. This dataset[5] contains 34,737 pairs of KUs. To ensure a fair comparison with the prior work [33], we use the same data for training, validation, and testing, containing 208,423, 34,737, and 104,211 pairs of KU, respectively. Our input data is the concatenation of two KUs. Specifically, each KU contains one question post and three corresponding answer posts. For the question post, we include the title, body, and code snippets. For the answer post, we consider the body and code.

Evaluation Metrics.
Following prior work [33], we adopt the micro-averaging method to calculate Micro-precision, Micro-recall, and Micro-F1 as evaluation metrics.

[4]https://huggingface.co/distilroberta-base
[5]https://anonymousaaai2019.github.io

Table 2. Experiment Results for Tag Recommendation Task

Group	Representation	P@1	R@1	F1@1	P@3	R@3	F1@3	P@5	R@5	F1@5
SOTA	**PTM4Tag**	0.875	0.875	0.875	0.586	0.756	0.641	0.417	0.805	0.526
SO-Specific	**Post2Vec**	0.875	0.875	0.875	0.585	0.754	0.639	0.416	0.804	0.525
	BERTOverflow	0.088	0.088	0.088	0.089	0.094	0.095	0.083	0.163	0.105
General Domain	**RoBERTa**	0.878	0.878	0.878	0.591	0.761	0.646	0.418	0.804	0.527
	Longformer	0.852	0.852	0.852	0.559	0.721	0.612	0.397	0.769	0.502
	GPT2	0.884	0.884	0.884	0.593	0.763	0.648	0.418	0.805	0.528
SE Domain	**CodeBERT**	0.876	0.876	0.876	0.588	0.758	0.642	0.418	0.805	0.527
	GraphCodeBERT	0.874	0.875	0.875	0.582	0.751	0.636	0.410	0.791	0.517
	seBERT	0.088	0.088	0.088	0.089	0.094	0.095	0.083	0.163	0.105
	CodeT5	0.887	0.887	0.887	0.599	0.770	0.653	0.420	0.809	0.530
	PLBart	0.883	0.883	0.883	0.600	0.773	0.656	0.422	0.811	0.532
	CodeGen	0.872	0.872	0.872	0.584	0.751	0.638	0.411	0.792	0.519
Our Model	**SOBERT**	**0.905**	**0.905**	**0.905**	**0.615**	**0.790**	**0.671**	**0.437(+3.4%)**	**0.836(+3.0%)**	**0.551(+3.4%)**

Table 3. Experimental Results for API Recommendation Task

Group	Representation	MRR	MAP	P@1	P@3	P@5	R@1	R@3	R@5
SOTA	**CLEAR**	0.739	0.753	0.482	0.560	0.562	0.629	0.766	0.793
SO-Specific	**Post2Vec**	0.735	0.745	0.471	0.560	0.556	0.625	0.774	0.801
	BERTOverflow	0.753	0.778	0.521	0.639	0.651	0.681	0.774	0.762
General domain	**RoBERTa**	0.777	0.790	0.537	0.640	0.653	0.689	0.782	0.815
	Longformer	0.767	0.782	0.525	0.623	0.646	0.683	0.772	0.793
	GPT2	0.766	0.782	0.528	0.641	0.650	0.683	0.772	0.795
SE domain	**CodeBERT**	0.781	0.800	0.564	0.641	0.659	0.712	0.772	0.793
	GraphCodeBERT	0.784	0.804	0.537	0.652	0.663	0.693	0.803	0.829
	seBERT	0.754	0.777	0.525	0.624	0.635	0.678	0.749	0.772
	CodeT5	0.779	0.796	0.544	0.643	0.651	0.693	0.786	0.809
	PLBart	0.762	0.782	0.521	0.619	0.633	0.679	0.768	0.795
	CodeGen	0.721	0.735	0.556	0.627	0.636	0.660	0.705	0.718
Our Model	**SOBERT**	**0.807(+2.9%)**	**0.826(+2.7%)**	**0.579**	**0.678**	**0.684**	**0.732**	**0.801**	**0.832**

Implementation Details.
We concatenate a pair of posts as the input to train a multi-class classifier. We fine-tuned Longformer on a sequence length of 1,024 and fine-tuned other pre-trained models on a sequence length of 512. For all experiments, we set the batch size as 32 and the epoch number as 5. We select the model with the smallest validation loss to run the evaluation.

5 EXPERIMENTAL RESULTS

This section describes the experiment results and answers our research questions. The experimental results are summarized in Table 2, 3, and 4, respectively.

RQ1: How effective are the existing Stack Overflow post representation models?

The experimental results of our tag recommendation experiments are summarized in Table 2. PTM4Tag achieves a performance of 0.417, 0.805, and 0.526 in terms of Precision@5, Recall@5, and F1-score@5, respectively. However, the extra inclusion of Post2Vec lowers the scores to 0.416, 0.804, and 0.525, respectively. In contrast, BERTOverflow struggles in the task with surprisingly low scores of 0.083, 0.163, and 0.105.

Table 4. Experiment Result for Relatedness Prediction Task

Group	Representation	F1-Score	Precision	Recall
SOTA	**ASIM**	0.785	0.785	0.785
SO-Specific	**Post2Vec**	0.768	0.768	0.768
	BERTOverflow	0.697	0.697	0.697
General Domain	**RoBERTa**	0.787	0.787	0.787
	Longformer	0.786	0.786	0.786
	GPT2	0.765	0.765	0.765
SE domain	**CodeBERT**	0.803	0.803	0.803
	GraphCodeBERT	0.801	0.801	0.801
	seBERT	0.799	0.799	0.799
	CodeT5	0.784	0.784	0.784
	PLBart	0.770	0.770	0.770
	CodeGen	0.765	0.765	0.765
Our Model	**SOBERT**	**0.825(+2.7%)**	**0.825(+2.7%)**	**0.825(+2.7%)**

For API recommendation, as shown in Table 3, combining Post2Vec with the state-of-the-art approach CLEAR also fails to boost the performance. CLEAR itself obtains an MRR score of 0.739 and MAP score of 0.753. Yet, with the integration of Post2Vec, these values diminish slightly to 0.735 and 0.745, respectively. Notably, BERTOverflow achieves scores of 0.753 in MRR and 0.778 in MAP.

In the relatedness prediction task, as detailed in Table 4, integrating Post2Vec with ASIM leads to a minor decrease in the F1-score, moving from 0.785 to 0.768. BERTOverflow, on the other hand, lags behind ASIM with an F1-score of 0.697.

Overall, Post2Vec can not enhance the performance of the state-of-the-art solutions across the evaluated downstream tasks. Furthermore, BERTOverflow demonstrates poor results in classification tasks and only achieves comparable performance with the state-of-the-art solution in API recommendation.

Answer to RQ1: The existing Stack Overflow representation methods fail to improve state-of-the-art performance in the three evaluated downstream tasks.

RQ2: How effective are the popular transformer-based language models for the targeted downstream tasks?

In the tag recommendation task, as demonstrated in Table 2, the state-of-the-art approach PTM4Tag is outperformed by numerous transformer-based pre-trained representation models. While the F1-score@5 of PTM4Tag is 0.526, PLBart achieves an F1-score@5 of 0.532, which makes it the best transformer-based pre-trained model. In contrast, seBERT significantly underperformed in this task, with an F1-score@5 of only 0.105.

Table 3 shows that CLEAR is no longer the best-performing method in API recommendation. Replacing the embedding of Distilled RoBERTa in the original design of CLEAR with other transformer-based language models increases the performance. Particularly, GraphCodeBERT boosts the performance of CLEAR by 3.8% and 5.0% in terms of MRR and MAP. For Precision@1,3,5 and Recall@1,3,5, GraphCodeBERT outperforms CLEAR by 6.7% to 22.0%. The worst representation model is CodeGen, which achieves MRR and MAP of 0.721 and 0.735. Meanwhile, both CodeBERT and GraphCodeBERT can surpass an MRR of 0.78.

The results of the relatedness prediction task are presented in Table 4. We observe that ASIM, the state-of-the-art technique in relatedness prediction, is outperformed by other transformer-based language models. While ASIM achieves a score of 0.785 in the F1-score, CodeBERT drives forward the state-of-the-art performance by

2.3% with an F1-score of 0.803. RoBERTa, GraphCodeBERT, Longformer, and seBERT have an F1-score of 0.787, 0.801, 0.786, and 0.799, all outperforming ASIM.

Overall, models like CodeBERT, RoBERTa, and GraphCodeBERT can consistently give promising representations in all three tasks, proving their generalizability and effectiveness in a wide range of SE-related tasks.

Answer to RQ2: Representations generated by CodeBERT, RoBERTa, and GraphCodeBERT consistently outperform each state-of-the-art technique from the targeted downstream tasks. However, none of the models can always be the best performer.

RQ3: Is further pre-training on Stack Overflow data helpful in building a better model?

Our experimental results show that there is no "one-size-fits-all" model in representing Stack Overflow posts, which could consistently outperform others in the considered tasks. Such a phenomenon delivers an intuition that there is an improvement opportunity in the representation technique for Stack Overflow. Based on the common practice that a second phase of in-domain pre-training leads to performance gains [17], we conduct additional pre-training for a transformer-based model (i.e., CodeBERT) with the Stack Overflow dataset. We name it SOBERT.

Pre-training Details: We have leveraged the Stack Overflow dump dated June 2023, which has 23 million question posts as the training corpus. The raw dataset has a size of approximately 70G. We also make sure that there is no overlapping between the test/validation datasets of three downstream tasks and the pre-training data of SOBERT. Many previous works have removed the code snippets of a Stack Overflow post during the pre-processing stage [26, 59]. According to the statistics conducted by Xu et al. [54], more than 70% of the Stack Overflow contains at least one code snippet. As a result, the removal of code snippets would result in losing a significant of information, and they should be considered to learn an effective post representation. As the code snippets within the body of a post are enclosed in HTML tags `<pre><code>` and `</code></pre>`, we cleaned the redundant HTML tags with regular expression `<pre><code>([\s\S]*?)<//code><//pre>`. As a result, the pre-training data of SOBERT contains the title, body, and code snippets of a Stack Overflow question post. We have initialized SOBERT based on the checkpoint of the CodeBERT model and pre-trained SOBERT using the MLM objective with a standard masking rate of 15%. The batch size is set as 256, and the learning rate is 1e-4. The training process takes around 100 hours for eight Nvidia V100 GPUs with 16 GB of memory to complete. The detailed code used is included in the replication package provided.

The experimental results show that SOBERT achieves the best performance for every downstream task. For tag recommendation, SOBERT achieves an F1-score@5 of 0.551 and beats PLBart by 3.4%; for API recommendation, SOBERT performs with 0.807 in terms of MRR and outperforms GraphCodeBERT by 2.9%.; and for relatedness prediction, it accomplishes an F1-score of 0.824 and outperforms CodeBERT by 2.7%.

We conduct the Wilcoxon Signed Rank [12] at a 95% significance level (i.e., p-value < 0.05) and calculate Cliff's delta [11] on the paired data corresponding to SOBERT and the best-performing competing representation model in each task (i.e., PLBart in tag recommendation, CodeBERT in relatedness prediction, and GraphCodeBERT in API recommendation). The significance test has been conducted on the values of evaluation metrics (F1-score@5 in tag recommendation, F1-score in relatedness prediction, MRR in API recommendation). For Cliff's delta, we consider delta that are less than 0.147, between 0.147 and 0.33, between 0.33 and 0.474, and above 0.474 as Negligible (N), Small (S), Medium (M), and Large (L) effect size, respectively following previous literature [11]. We observe that SOBERT significantly (p-value < 0.05) and substantially (Cliff's delta is 0.31–0.55) outperforms the comparing model.

Answer to RQ3: Further pre-training with the Stack Overflow data yields better representation in modelling Stack Overflow posts. SOBERT consistently achieves state-of-the-art performance in all the targeted downstream tasks.

6 DISCUSSION

6.1 Lessons Learned

Lesson #1*. *Incorporating post embeddings from an external approach does not boost the performance of neural network models.

Xu et al. [54] demonstrated that appending the distributed post representation learned by Post2Vec to the manually crafted feature vector can increase the performance of traditional machine learning algorithms, for example, Support Vector Machine [55] and Random Forest [5], in a set of Stack Overflow-related tasks. However, these benefits are not observed for the state-of-the-art techniques that are based on deep neural networks. This is potentially caused by the design of neural networks that automatically extract feature vectors and continuously optimize the representations. It indicates that deep neural networks may lose the effectiveness of external embeddings while optimizing the parameters of the feature extractor.

Lesson #2*. *Models with broader background knowledge derive better results than those with specific knowledge.

Intuitively, BERTOverflow is expected to produce the desired Stack Overflow post representation as it is specifically designed for Stack Overflow data. A major difference between BERTOverflow and others is the vocabulary. As BERTOverflow is pre-trained from scratch with the Stack Overflow data, its vocabulary should be more suitable for modeling Stack Overflow posts than general domain models.

Surprisingly, our experiment results show that other transformer-based language models outperform BERTOverflow by a substantial margin across all three tasks. It gives an extremely poor performance in the tag recommendation task. By inspecting the prediction results of BERTOverflow in the tag prediction task, we notice that the top-5 predictions made by BERTOverflow are always the most frequent tags ('python', 'java', 'c#', 'java-script', and 'android') from the dataset. We observe seBERT has a similar performance as BERTOverflow in the tag recommendation task.

We hypothesize that the poor performance of BERTOverflow and seBERT is because these models lack a sufficient amount of pre-training to perform well. Since seBERT and BERTOverflow are trained from scratch, they require much more pre-training effort than continued pre-training with existing models. To prove this concept, we perform additional pre-training on BERTOverflow with the same dataset as SOBERT. The further pre-training was done with the same hyper-parameters as SOBERT, and it took 23 hours on four 16GB Nvidia V100 GPUs to complete. We denote this new model as BERTOverflow$_{\text{NEW}}$.[6]

To independently evaluate the effect of domain-specific vocabulary, we train additional two transformer-based models from scratch. Both models contain 12 layers of encoder modules, but one with the vocabulary of BERTOverflow and the other with the vocabulary of RoBERTa. The pre-training process is the same as for BERTOverflow$_{\text{NEW}}$. We refer to these two models as BERTOverflow$_{\text{vocab}}$, and RoBERT$_{\text{vocab}}$. From Table 8, we observe the significant performance improvements of BERTOverflow$_{\text{NEW}}$ compared to BERTOverflow. We also observe that the performance of BERTOverflow$_{\text{vocab}}$ and RoBERTa$_{\text{vocab}}$ are very similar to each other. Overall, our experiments have shown that domain-specific vocabulary has a negligible effect on all three tasks. The more important factor tends to be the amount of pre-training. Moreover, pre-training from scratch is commonly

[6]Please note that we could not apply further pre-training to seBERT due to the constraints of limited resources to handle BERT$_{\text{Large}}$ architecture.

Table 5. Comparison of the results from different tokenizers on the dataset of tag recommendation.

Model	Mean	Min	25%	50%	75%	Max	Longer than 512
CodeT5	253.3	8	93	158	278	27449	9.3%
RoBERTa	254.7	9	92	157	279	27437	9.4%
PLBart	239	8	90	152	266	19467	8.4%
BERTOverflow	250	8	90	156	278	27596	9.3%
seBERT	255.4	8	92	158	283	27600	9.7%

Table 6. Comparison of the results from different tokenizers on the dataset of API recommendation.

Model	Mean	Min	25%	50%	75%	Max
CodeT5	11.0	4	8	10	14	53
RoBERTa	11.3	4	8	11	14	49
PLBart	10.5	3	7	10	13	51
BERTOverflow	10.3	4	7	10	13	48
seBERT	10.6	4	7	10	13	48

Table 7. Comparison of the results from different tokenizers on the dataset of relatedness prediction.

Model Name	Mean	Min	25%	50%	75%	Max	Longer than 512
CodeT5	1826.6	102	919	1407	2213	33989	94.6%
RoBERTa	1897.7	99	944	1454	2301	34540	94.8%
PLBart	1717.9	96	877	1340	2090	32995	93.7%
BERTOverflow	1862.7	98	937	1444	2264	41346	94.7%
seBERT	1917.4	98	961	1485	2331	41750	95.0%

considered an expensive process. Initializing new representation models based on the checkpoint of a popular model reduces the risk, and the tag recommendation task is a good indicator to demonstrate the generalizability and the sufficiency of pre-training of transformer-based representation models.

For a more comprehensive insight, we analyze the lengths of tokenized text for different representation models on the datasets used in this paper. Table 5, 6, and 7 show the statistics for the lengths of different tokenizers. Given that representation models like CodeBERT, GraphCodeBERT, Longformer, GPT2, and CodeGen share the same tokenizer as RoBERTa, we omit these representation models from this analysis. We notice that PLBart's tokenizer can generate the shortest tokenizations across all datasets. Interestingly, even though BERTOverflow and seBERT are being developed with SE-specific vocabularies, the lengths of their tokenized text are almost the same as RoBERTa's. For example, BERTOverflow's average tokenization length in the tag recommendation task is 250, while RoBERTa's is slightly higher at 254.7.

Lesson #3**. **Despite considering a longer input length, Longformer does not produce better representations for posts.

Conventional transformer-based models like CodeBERT and RoBERTa cannot handle long sequences due to the quadratic complexity of the self-attention mechanism [48] and accept a maximum of 512 tokens as the input. From Table 5, 6, and 7, we can observe that the ratios of data that are longer than 512 tokens are approximately 9%, 0%, and 94% in tag recommendation, API recommendation, and relatedness prediction, respectively.

Table 8. Results for variants of BERTOverflow, RoBERTa, and Longformer

	Tag Recommendation			API Recommendation		Relatedness Prediction		
	P@5	**R@5**	**F1@5**	**MRR**	**MAP**	**P**	**R**	**F1**
BERTOverflow	0.083	0.163	0.105	0.753	0.778	0.697	0.697	0.697
BERTOverflow-New	0.411	0.791	0.519	0.779	0.793	0.789	0.789	0.789
BERTOverflow-vocab	0.411	0.790	0.518	0.771	0.785	0.788	0.788	0.788
RoBERTa-vocab	0.412	0.794	0.520	0.778	0.792	0.793	0.793	0.793
Longformer-512	0.397	0.768	0.502	0.768	0.783	0.785	0.785	0.785
Longformer-1024	0.397	0.769	0.502	0.767	0.782	0.786	0.786	0.786

Table 9. Examples of predictions made by CodeBERT and SOBERT in the tag recommendation task

Post ID	**Post Title**	**CodeBERT Tag Prediction**	**SOBERT Tag Prediction**	**True Tag**
13202867	Fixed size of Flexslider	‘apache-flex’, ‘frameworks’, ‘ios’, ‘swift’, ‘xcode’	‘css‘’, ‘html‘’, ‘image‘’, ‘javascript’, ‘jquery’	‘css’, ‘html’, ‘javascript’
30434343	What is the right way to typecheck dependent lambda abstraction using ‘bound’?	‘.net’, ‘binding’, ‘c#’, ‘lambda’, ‘type-inference’	‘functional-programming’, ‘haskell’, ‘lambda’, ‘type-inference’, ‘types’	‘haskell’
17849870	Closed type classes	‘.net’, ‘c++’, ‘d’, ‘f#’, ‘performance’	‘applicative‘’, ‘ghc‘’, ‘haskell‘’, ‘typeclass‘, types‘	‘haskell’, ‘static-analysis’, ‘typeclass’, ‘types’

In Table 6, we can see that the dataset of API recommendation has a short length, where the longest tokenized text has a length of 53. This is because only the title of a post is considered in this task. As Longformer is implemented with a simplified attention mechanism (introduced in Section 2), which only gives its advantage in handling long text, this explains why CodeBERT, and RoBERTa outperform Longformer in API recommendation.

From Table 5 and 7, we can see that both dataset contains data samples that are longer than the 512 limit. Especially in related prediction (Table 7), the average length of each knowledge unit (KU) is more than 1800 tokens. The lengthy text is because each KU consists of question posts and a set of corresponding answer posts. Surprisingly, Longformer fails to perform better than the other model that belongs to the general domain (i.e., RoBERTa) as well as models from the SE domain, even though it takes much longer input in this task.

We further compare the performance of Longformer by varying the input size considering the first 512 and 1,024 tokens. The additional experimental results are shown in Table 8. These additional settings do not differ in performance. It indicates that diversifying the input size does not affect Longformer's performance on post representation. A potential interpretation would be the important features for representing Stack Overflow posts lie in the first part of each post (e.g., Title serves as a succinct summary of the post). It is not worth trying Longformer unless one strictly needs the entire content of Stack Overflow posts.

Lesson #4**. **We advocate future studies related to Stack Overflow consider the SOBERT as the underlying baseline.

Our experiment results demonstrate that further pre-training based on in-domain data leads to better Stack Overflow post representation. By initializing SOBERT with the CodeBERT checkpoint and performing further pre-training on Stack Overflow data, we have noticed that SOBERT consistently outperforms the original CodeBERT and produces new state-of-the-art performance for all three tasks.

In Table 9, we present three examples of the prediction results of CodeBERT and SOBERT for the tag recommendation task. We observe that CodeBERT is making wrong predictions like ".net" and "c#" when the question is about "haskell", while SOBERT is capable of making the correct predictions. CodeBERT may lack knowledge of programming languages like Haskell and Lua since it is pre-trained on artifacts from Python, Java, JavaScript, PHP, Ruby, and Go. Taking the Stack Overflow post with ID 13202867 as another example, the question is about Flexslider, a jQuery slider plugin. In the given example, SOBERT could successfully make connections to tags like 'jQuery' and 'css' while CodeBERT struggles to give meaningful predictions.

Overall, by continuing the pre-training process on Stack Overflow data, SOBERT outperforms CodeBERT in three popular Stack Overflow-related tasks. We advocate future studies to consider SOBERT as their underlying baseline. To facilitate the usage of the SOBERT proposed in this work, we plan to release it to HuggingFace[7] so that it can be used by simply calling the interface.

6.2 Threats to Validity

Threats to internal validity. To ensure the correct implementation of the baseline methods (i.e., Post2Vec, PTM4Tag, CLEAR, and ASIM), we reused the replication package released by the original authors.[8,9,10,11] When investigating the effectiveness of various pre-trained models, we used the implementation of each from the popular open-source community *HuggingFace*. Another threat to the internal validity is the hyperparameter setting we used to pre-train SOBERT and fine-tune the representation models. To mitigate this threat, the leveraged hyper-parameters in this paper in both the pre-training and fine-tuning phases were reported in other prior reputable literature as recommended or optimal [14, 28, 52].
Threats to external validity. One threat to external validity relates our results may not generalize to those newly emerging topics or other Stack Overflow-related downstream tasks. We have minimized this threat by considering multiple downstream tasks.
Threats to construct validity. We reuse the same evaluation metrics in our baseline methods [18, 33, 52]. To further reduce the risk, we conduct the Wilcoxon signed-rank statistical hypothesis test and Cliff's delta to check whether the output between the two competing approaches is significant and substential.

7 RELATED WORK

In this section, we review two lines of research that most relate to our work: pre-trained models for SE and mining Stack Overflow posts.

7.1 Pre-trained Models for Software Engineering

Inspired by the success of transformer-based pre-trained models in NLP, there is an increasing research interest in exploring pre-training tasks and applying pre-trained models for SE [9, 23, 27, 29–31, 47, 50, 58].

One set of research focuses on learning semantic and contextual representations of source code; after pre-training, these models can be fine-tuned to solve SE downstream tasks. ContraCode [23] is an encoder-only model that uses a contrastive pre-training task to learn code functionality. It classifies JavaScript programs into positive pairs (i.e., functionally similar) and negative pairs (i.e., functionally dissimilar). During the contrastive pre-training, query programs are used to retrieve positive programs. Positive programs are pushed together, while negative programs are pushed apart. CodeGPT [29] is a pure decoder model trained in programming languages. It leverages the Python and Java corpora from the CodeSearchNet dataset [22].

[7]https://huggingface.co/
[8]https://github.com/maxxbw54/Post2Vec
[9]https://github.com/soarsmu/PTM4Tag
[10]https://github.com/Moshiii/CLEAR-replication
[11]https://github.com/Anonymousmsr/ASIM

Another set of research focuses on leveraging the transformer-based model to automate SE challenges [9, 27, 30, 31, 47, 58]. Zhang et al. [58] conduct a comparative study on transformer-based pre-trained models with prior SE-specific tools in sentiment analysis for SE. The experimental results show that transformer-based pre-trained models is more ready for real use than the prior tools. Lin et al. [27] find that BERT can boost the performance of traceability tasks in open-source projects. They investigate three BERT architectures, i.e., Single-BERT, Siamese-BERT, and Twin-BERT. The results indicate that the single-BERT can generate the most accurate links, while a Siamese-BERT architecture produced comparable effectiveness with significantly better efficiency. Ciniselli et al. [9] investigate the potential of transformer-based models in the code completion task, ranging from single token prediction to the prediction of the entire code blocks. Experiments are conducted on several variants of two popular transformer-based models, namely RoBERTa and T5. The experimental results demonstrate that T5 is the most effective in supporting code completion. T5 variants can achieve prediction accuracies up to 29% for whole block prediction and 69% for token-level prediction. Mastropaolo et al. [31] further explore the effusiveness of T5 for bug fixing, injecting code mutants, generating assert statements, and code summarization. The study finds that the T5 can outperform the previous state-of-the-art deep learning-based approaches for those tasks. Tufano et al. [47] perform an empirical evaluation of the T5 model in automating the code review process. The experiments are performed on a much larger and realistic code review dataset, and the T5-based model outperforms previous deep learning models in this task. Mastropaolo et al. [30] present LANCE, a log statement recommendation system using the transformer architecture. LANCE can accurately determine where to place a log statement at 65.9% of the time, select the correct log level at 66.2% of the time, and generate a fully accurate log statement with a relevant message in 15.2% of instances.

Different from these works, we focus on a comprehensive set of Stack Overflow-related tasks in this paper. In addition to fine-tuning the transformer-based pre-trained representation models, we also further pre-trained SOBERT on Stack Overflow data.

7.2 Mining Stack Overflow Posts

We address tag recommendation [18, 54], API recommendation [6, 52], and relatedness prediction [33, 55] in this work. Others also explored other tasks for mining Stack Overflow posts to support software developers, such as post recommendation [41], multi-answer summarization [56], and controversial discussions [40].

Rubei et al. [41] propose an approach named PostFinder, which aims to retrieve Stack Overflow posts that are relevant to API function calls that have been invoked. They make use of Apache Lucene to index the textual content and code in Stack Overflow to improve efficiency. In both the data collection and query phase, they make use of the data available at hand to optimize the search process. Specifically, they retrieve and augment posts with additional data to make them more exposed to queries. Besides, they boost the context code to construct a query that contains the essential information to match the stored indexes.

Xu et al. [56] investigate the multi-answer posts summarization task for a given input question, which aims to help developers get the key points of several answer posts before they dive into the details of the results. They propose an approach *AnswerBot*, which contains three main steps, i.e., relevant question retrieval, useful answer paragraph selection, and diverse answer summary generation.

Ren et al. [40] investigate the controversial discussions in Stack Overflow. They find that there is a large scale of controversies in Stack Overflow, which indicates that many answers are wrong, less optimal, and out-of-date. Our work and their work are complementary to each other, and all aim to boost automation in understanding and utilizing Stack Overflow contents.

8 CONCLUSION AND FUTURE WORK

In this paper, we empirically study the effectiveness of varying techniques for modeling Stack Overflow posts, including approaches that are specially designed for Stack Overflow posts (i.e., Post2Vec and BERTOverflow), SE domain representation models (i.e., CodeBERT, GraphCodeBERT, seBERT, CodeT5, PLBart, CodeGen) and general domain representation models (i.e., RoBERTa, LongFormer, and GPT2). We evaluate the performance of these representation models on three popular and representative Stack Overflow-related tasks, which are tag recommendation, API recommendation, and relatedness prediction.

Our experimental results show that Post2Vec is unable to enhance the representations that are automatically extracted by deep learning-based methods and BERTOverflow performs surprisingly worse than other transformer-based language models. Furthermore, there does not exist one representation technique that could consistently outperform other representation models. Our findings indicate the current research gap in representing Stack Overflow posts. Thus, we propose SOBERT with a simple-yet-effective strategy. We pre-train SOBERT with the posts from Stack Overflow. As a result, SOBERT improves the performance of the original CodeBERT and consistently outperforms other models on all three tasks, confirming that further pre-training on Stack Overflow data helps build Stack Overflow representation.

In the future, we would also extend our research to other SQA sites, such as AskUbuntu[12]. Moreover, we would also consider other Stack Overflow-related downstream tasks into account in the future.

9 DATA AVAILABILITY

The replication package of the data and code used in this paper is available at **https://figshare.com/s/7f80db836305607b89f3**.

10 ACKNOWLEDGEMENT

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 author(s) and do not reflect the views of National Research Foundation, Singapore.

REFERENCES

[1] Md Ahasanuzzaman, Muhammad Asaduzzaman, Chanchal K Roy, and Kevin A Schneider. 2018. Classifying stack overflow posts on API issues. In *2018 IEEE 25th international conference on software analysis, evolution and reengineering (SANER)*. IEEE, 244–254.

[2] Wasi Uddin Ahmad, Saikat Chakraborty, Baishakhi Ray, and Kai-Wei Chang. 2021. Unified Pre-training for Program Understanding and Generation. In *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*, Kristina Toutanova, Anna Rumshisky, Luke Zettlemoyer, Dilek Hakkani-Tür, Iz Beltagy, Steven Bethard, Ryan Cotterell, Tanmoy Chakraborty, and Yichao Zhou (Eds.). Association for Computational Linguistics, 2655–2668. https://doi.org/10.18653/v1/2021.naacl-main.211

[3] Shirani Amirreza, X Bowen, L David, Thamar Solorio, and Amin Alipour. 2019. Question relatedness on stack overflow: the task, dataset, and corpus-inspired models. In *Proceedings of the AAAI Reasoning for Complex Question Answering Workshop*.

[4] Iz Beltagy, Matthew E. Peters, and Arman Cohan. 2020. Longformer: The Long-Document Transformer. *CoRR* abs/2004.05150 (2020). arXiv:2004.05150 https://arxiv.org/abs/2004.05150

[5] Stefanie Beyer, Christian Macho, Massimiliano Di Penta, and Martin Pinzger. 2018. Automatically classifying posts into question categories on stack overflow. In *2018 IEEE/ACM 26th International Conference on Program Comprehension (ICPC)*. IEEE, 211–21110.

[6] Liang Cai, Haoye Wang, Qiao Huang, Xin Xia, Zhenchang Xing, and David Lo. 2019. BIKER: a tool for Bi-information source based API method recommendation. In *Proceedings of the ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/SIGSOFT FSE 2019, Tallinn, Estonia, August 26-30, 2019*, Marlon Dumas, Dietmar Pfahl, Sven Apel, and Alessandra Russo (Eds.). ACM, 1075–1079. https://doi.org/10.1145/3338906.3341174

[12]https://askubuntu.com/

[7] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Pondé de Oliveira Pinto, Jared Kaplan, Harrison Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Joshua Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. 2021. Evaluating Large Language Models Trained on Code. *CoRR* abs/2107.03374 (2021). arXiv:2107.03374 https://arxiv.org/abs/2107.03374

[8] Xinyun Chen, Chang Liu, and Dawn Song. 2018. Tree-to-tree Neural Networks for Program Translation. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett (Eds.). 2552–2562. https://proceedings.neurips.cc/paper/2018/hash/d759175de8ea5b1d9a2660e45554894f-Abstract.html

[9] Matteo Ciniselli, Nathan Cooper, Luca Pascarella, Antonio Mastropaolo, Emad Aghajani, Denys Poshyvanyk, Massimiliano Di Penta, and Gabriele Bavota. 2022. An Empirical Study on the Usage of Transformer Models for Code Completion. *IEEE Trans. Software Eng.* 48, 12 (2022), 4818–4837. https://doi.org/10.1109/TSE.2021.3128234

[10] Kevin Clark, Minh-Thang Luong, Quoc V. Le, and Christopher D. Manning. 2020. ELECTRA: Pre-training Text Encoders as Discriminators Rather Than Generators. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net. https://openreview.net/forum?id=r1xMH1BtvB

[11] Norman Cliff. 2014. *Ordinal methods for behavioral data analysis*. Psychology Press.

[12] William Jay Conover. 1999. *Practical nonparametric statistics*. Vol. 350. john wiley & sons.

[13] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, Jill Burstein, Christy Doran, and Thamar Solorio (Eds.). Association for Computational Linguistics, 4171–4186. https://doi.org/10.18653/v1/n19-1423

[14] Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, and Ming Zhou. 2020. CodeBERT: A Pre-Trained Model for Programming and Natural Languages. In *Findings of the Association for Computational Linguistics: EMNLP 2020, Online Event, 16-20 November 2020 (Findings of ACL, Vol. EMNLP 2020)*, Trevor Cohn, Yulan He, and Yang Liu (Eds.). Association for Computational Linguistics, 1536–1547. https://doi.org/10.18653/v1/2020.findings-emnlp.139

[15] Xiaodong Gu, Hongyu Zhang, D. Zhang, and Sunghun Kim. 2016. Deep API learning. *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering* (2016).

[16] Daya Guo, Shuo Ren, Shuai Lu, Zhangyin Feng, Duyu Tang, Shujie Liu, Long Zhou, Nan Duan, Alexey Svyatkovskiy, Shengyu Fu, Michele Tufano, Shao Kun Deng, Colin B. Clement, Dawn Drain, Neel Sundaresan, Jian Yin, Daxin Jiang, and Ming Zhou. 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

[17] Suchin Gururangan, Ana Marasović, Swabha Swayamdipta, Kyle Lo, Iz Beltagy, Doug Downey, and Noah A Smith. 2020. Don't stop pretraining: adapt language models to domains and tasks. *arXiv preprint arXiv:2004.10964* (2020).

[18] Junda He, Bowen Xu, Zhou Yang, DongGyun Han, Chengran Yang, and David Lo. 2022. PTM4Tag: Sharpening Tag Recommendation of Stack Overflow Posts with Pre-trained Models. *CoRR* abs/2203.10965 (2022). https://doi.org/10.48550/arXiv.2203.10965 arXiv:2203.10965

[19] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation* 9, 8 (1997), 1735–1780.

[20] Qiao Huang, Xin Xia, Zhenchang Xing, D. Lo, and Xinyu Wang. 2018. API Method Recommendation without Worrying about the Task-API Knowledge Gap. *2018 33rd IEEE/ACM International Conference on Automated Software Engineering (ASE)* (2018), 293–304.

[21] Qiao Huang, Xin Xia, Zhenchang Xing, David Lo, and Xinyu Wang. 2018. API method recommendation without worrying about the task-API knowledge gap. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering, ASE 2018, Montpellier, France, September 3-7, 2018*, Marianne Huchard, Christian Kästner, and Gordon Fraser (Eds.). ACM, 293–304. https://doi.org/10.1145/3238147.3238191

[22] Hamel Husain, Ho-Hsiang Wu, Tiferet Gazit, Miltiadis Allamanis, and Marc Brockschmidt. 2019. CodeSearchNet Challenge: Evaluating the State of Semantic Code Search. *CoRR* abs/1909.09436 (2019). arXiv:1909.09436 http://arxiv.org/abs/1909.09436

[23] Paras Jain, Ajay Jain, Tianjun Zhang, Pieter Abbeel, Joseph Gonzalez, and Ion Stoica. 2021. Contrastive Code Representation Learning. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, EMNLP 2021, Virtual Event / Punta Cana, Dominican Republic, 7-11 November, 2021*, Marie-Francine Moens, Xuanjing Huang, Lucia Specia, and Scott Wen-tau Yih (Eds.). Association for Computational Linguistics, 5954–5971. https://doi.org/10.18653/v1/2021.emnlp-main.482

[24] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. 2015. Deep learning. *nature* 521, 7553 (2015), 436–444.

[25] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. 2020. BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*, Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel R. Tetreault (Eds.). Association for Computational Linguistics, 7871–7880. https://doi.org/10.18653/v1/2020.acl-main.703

[26] Can Li, Ling Xu, Meng Yan, and Yan Lei. 2020. TagDC: A tag recommendation method for software information sites with a combination of deep learning and collaborative filtering. *J. Syst. Softw.* 170 (2020), 110783. https://doi.org/10.1016/j.jss.2020.110783

[27] Jinfeng Lin, Yalin Liu, Qingkai Zeng, Meng Jiang, and Jane Cleland-Huang. 2021. Traceability Transformed: Generating more Accurate Links with Pre-Trained BERT Models. In *43rd IEEE/ACM International Conference on Software Engineering, ICSE 2021, Madrid, Spain, 22-30 May 2021*. IEEE, 324–335. https://doi.org/10.1109/ICSE43902.2021.00040

[28] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. RoBERTa: A Robustly Optimized BERT Pretraining Approach. *CoRR* abs/1907.11692 (2019). arXiv:1907.11692 http://arxiv.org/abs/1907.11692

[29] Shuai Lu, Daya Guo, Shuo Ren, Junjie Huang, Alexey Svyatkovskiy, Ambrosio Blanco, Colin Clement, Dawn Drain, Daxin Jiang, Duyu Tang, et al. 2021. Codexglue: A machine learning benchmark dataset for code understanding and generation. *arXiv preprint arXiv:2102.04664* (2021).

[30] Antonio Mastropaolo, Luca Pascarella, and Gabriele Bavota. 2022. Using Deep Learning to Generate Complete Log Statements. In *44th IEEE/ACM 44th International Conference on Software Engineering, ICSE 2022, Pittsburgh, PA, USA, May 25-27, 2022*. ACM, 2279–2290. https://doi.org/10.1145/3510003.3511561

[31] Antonio Mastropaolo, Simone Scalabrino, Nathan Cooper, David Nader-Palacio, Denys Poshyvanyk, Rocco Oliveto, and Gabriele Bavota. 2021. Studying the Usage of Text-To-Text Transfer Transformer to Support Code-Related Tasks. In *43rd IEEE/ACM International Conference on Software Engineering, ICSE 2021, Madrid, Spain, 22-30 May 2021*. IEEE, 336–347. https://doi.org/10.1109/ICSE43902.2021.00041

[32] Erik Nijkamp, Bo Pang, Hiroaki Hayashi, Lifu Tu, Huan Wang, Yingbo Zhou, Silvio Savarese, and Caiming Xiong. 2023. CodeGen: An Open Large Language Model for Code with Multi-Turn Program Synthesis. *ICLR* (2023).

[33] Jiayan Pei, Yimin Wu, Zishan Qin, Yao Cong, and Jingtao Guan. 2021. Attention-based model for predicting question relatedness on Stack Overflow. *2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR)* (2021), 97–107.

[34] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep Contextualized Word Representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2018, New Orleans, Louisiana, USA, June 1-6, 2018, Volume 1 (Long Papers)*, Marilyn A. Walker, Heng Ji, and Amanda Stent (Eds.). Association for Computational Linguistics, 2227–2237. https://doi.org/10.18653/v1/n18-1202

[35] Luca Ponzanelli, Andrea Mocci, Alberto Bacchelli, Michele Lanza, and David Fullerton. 2014. Improving Low Quality Stack Overflow Post Detection. In *30th IEEE International Conference on Software Maintenance and Evolution, Victoria, BC, Canada, September 29 - October 3, 2014*. IEEE Computer Society, 541–544. https://doi.org/10.1109/ICSME.2014.90

[36] Chen Qu, Liu Yang, Minghui Qiu, W. Bruce Croft, Yongfeng Zhang, and Mohit Iyyer. 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, SIGIR 2019, Paris, France, July 21-25, 2019*, Benjamin Piwowarski, Max Chevalier, Éric Gaussier, Yoelle Maarek, Jian-Yun Nie, and Falk Scholer (Eds.). ACM, 1133–1136. https://doi.org/10.1145/3331184.3331341

[37] Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. 2018. Improving language understanding by generative pre-training. (2018).

[38] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. 2019. Language models are unsupervised multitask learners. *OpenAI blog* 1, 8 (2019), 9.

[39] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *The Journal of Machine Learning Research* 21, 1 (2020), 5485–5551.

[40] Xiaoxue Ren, Zhenchang Xing, Xin Xia, Guoqiang Li, and Jianling Sun. 2019. Discovering, explaining and summarizing controversial discussions in community q&a sites. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 151–162.

[41] Riccardo Rubei, Claudio Di Sipio, Phuong Thanh Nguyen, Juri Di Rocco, and Davide Di Ruscio. 2020. PostFinder: Mining Stack Overflow posts to support software developers. *Inf. Softw. Technol.* 127 (2020), 106367. https://doi.org/10.1016/j.infsof.2020.106367

[42] Mike Schuster and Kuldip K Paliwal. 1997. Bidirectional recurrent neural networks. *IEEE transactions on Signal Processing* 45, 11 (1997), 2673–2681.

[43] Chi Sun, Luyao Huang, and Xipeng Qiu. 2019. Utilizing BERT for Aspect-Based Sentiment Analysis via Constructing Auxiliary Sentence. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, Jill Burstein, Christy Doran, and

Thamar Solorio (Eds.). Association for Computational Linguistics, 380–385. https://doi.org/10.18653/v1/n19-1035

[44] Jeniya Tabassum, Mounica Maddela, Wei Xu, and Alan Ritter. 2020. Code and Named Entity Recognition in StackOverflow. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, Online, 4913–4926. https://doi.org/10.18653/v1/2020.acl-main.443

[45] Henry Tsai, Jason Riesa, Melvin Johnson, Naveen Arivazhagan, Xin Li, and Amelia Archer. 2019. Small and Practical BERT Models for Sequence Labeling. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, EMNLP-IJCNLP 2019, Hong Kong, China, November 3-7, 2019*, Kentaro Inui, Jing Jiang, Vincent Ng, and Xiaojun Wan (Eds.). Association for Computational Linguistics, 3630–3634. https://doi.org/10.18653/v1/D19-1374

[46] Michele Tufano, Cody Watson, Gabriele Bavota, Massimiliano Di Penta, Martin White, and Denys Poshyvanyk. 2019. An Empirical Study on Learning Bug-Fixing Patches in the Wild via Neural Machine Translation. *ACM Trans. Softw. Eng. Methodol.* 28, 4 (2019), 19:1–19:29. https://doi.org/10.1145/3340544

[47] Rosalia Tufano, Simone Masiero, Antonio Mastropaolo, Luca Pascarella, Denys Poshyvanyk, and Gabriele Bavota. 2022. Using Pre-Trained Models to Boost Code Review Automation. In *44th IEEE/ACM 44th International Conference on Software Engineering, ICSE 2022, Pittsburgh, PA, USA, May 25-27, 2022*. ACM, 2291–2302. https://doi.org/10.1145/3510003.3510621

[48] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett (Eds.). 5998–6008. https://proceedings.neurips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html

[49] Julian Von der Mosel, Alexander Trautsch, and Steffen Herbold. 2022. On the validity of pre-trained transformers for natural language processing in the software engineering domain. *IEEE Transactions on Software Engineering* (2022), 1–1. https://doi.org/10.1109/TSE.2022.3178469

[50] Yue Wang, Weishi Wang, Shafiq R. Joty, and Steven C. H. Hoi. 2021. CodeT5: Identifier-aware Unified Pre-trained Encoder-Decoder Models for Code Understanding and Generation. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, EMNLP 2021, Virtual Event / Punta Cana, Dominican Republic, 7-11 November, 2021*, Marie-Francine Moens, Xuanjing Huang, Lucia Specia, and Scott Wen-tau Yih (Eds.). Association for Computational Linguistics, 8696–8708. https://doi.org/10.18653/v1/2021.emnlp-main.685

[51] Huihui Wei and Ming Li. 2017. Supervised Deep Features for Software Functional Clone Detection by Exploiting Lexical and Syntactical Information in Source Code. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017*, Carles Sierra (Ed.). ijcai.org, 3034–3040. https://doi.org/10.24963/ijcai.2017/423

[52] Moshi Wei, Nima Shiri Harzevili, Junjie Wang Yuchao Huang, and Song Wang. 2022. CLEAR: Contrastive Learning for API Recommendation. *44th International Conference on Software Engineering (ICSE)* (2022).

[53] Xin Xia, Lingfeng Bao, D. Lo, Pavneet Singh Kochhar, A. Hassan, and Zhenchang Xing. 2017. What do developers search for on the web? *Empirical Software Engineering* 22 (2017), 3149–3185.

[54] Bowen Xu, Thong Hoang, Abhishek Sharma, Chengran Yang, Xin Xia, and David Lo. 2021. Post2Vec: Learning Distributed Representations of Stack Overflow Posts. *IEEE Transactions on Software Engineering* (2021), 1–1. https://doi.org/10.1109/TSE.2021.3093761

[55] Bowen Xu, Amirreza Shirani, David Lo, and Mohammad Amin Alipour. 2018. Prediction of relatedness in stack overflow: deep learning vs. SVM: a reproducibility study. In *Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, ESEM 2018, Oulu, Finland, October 11-12, 2018*, Markku Oivo, Daniel Méndez Fernández, and Audris Mockus (Eds.). ACM, 21:1–21:10. https://doi.org/10.1145/3239235.3240503

[56] Bowen Xu, Zhenchang Xing, Xin Xia, and David Lo. 2017. AnswerBot: Automated generation of answer summary to developers' technical questions. In *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 706–716.

[57] Chengran Yang, Bowen Xu, Junaed Younus Khan, Gias Uddin, Donggyun Han, Zhou Yang, and David Lo. 2022. Aspect-Based API Review Classification: How Far Can Pre-Trained Transformer Model Go?. In *2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER). IEEE Computer Society.*

[58] Ting Zhang, Bowen Xu, Ferdian Thung, Stefanus Agus Haryono, David Lo, and Lingxiao Jiang. 2020. Sentiment analysis for software engineering: How far can pre-trained transformer models go?. In *2020 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 70–80.

[59] Pingyi Zhou, Jin Liu, Xiao Liu, Zijiang Yang, and John C. Grundy. 2019. Is deep learning better than traditional approaches in tag recommendation for software information sites? *Inf. Softw. Technol.* 109 (2019), 1–13. https://doi.org/10.1016/j.infsof.2019.01.002