Good things come in three: Generating SO Post Titles with Pre-Trained Models, Self Improvement and Post Ranking

Anonymous Author(s)

ABSTRACT

Stack Overflow is a prominent Q&A forum, supporting developers in seeking suitable resources on programming-related matters. Having high-quality question titles is an effective means to attract developers' attention. Unfortunately, this is often underestimated, leaving room for improvement. Research has been conducted, predominantly leveraging pre-trained models to generate titles from code snippets and problem descriptions. Yet, getting high-quality titles is still a challenging task, attributed to both the quality of the input data (e.g., containing noise and ambiguity) and inherent constraints in sequence generation models.

In this paper, we present `FILLER` as a solution to generating Stack Overflow post titles using a fine-tuned language model with self-improvement and post ranking. Our study focuses on enhancing pre-trained language models for generating titles for Stack Overflow posts, employing a training and subsequent fine-tuning paradigm for these models. To this end, we integrate the model's predictions into the training process, enabling it to learn from its errors, thereby lessening the effects of exposure bias. Moreover, we apply a post-ranking method to produce a variety of sample candidates, subsequently selecting the most suitable one. To evaluate `FILLER`, we perform experiments using benchmark datasets, and the empirical findings indicate that our model provides high-quality recommendations. Moreover, it significantly outperforms all the baselines, including `Code2Que`, `SOTitle`, `CCBERT`, `M3NSCT5`, and `GPT3.5-turbo`. A user study also shows that `FILLER` provides more relevant titles, with respect to `SOTitle` and `GPT3.5-turbo`.

KEYWORDS

Pre-trained models, Stack Overflow, Post Ranking

ACM Reference Format:
Anonymous Author(s). 2024. Good things come in three: Generating SO Post Titles with Pre-Trained Models, Self Improvement and Post Ranking. In *Proceedings of The 18th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM 2024).* ACM, New York, NY, USA, 12 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

When working on software projects, developers usually seek support from different sources to complete their programming tasks [29]. Among others, Stack Overflow (SO) is a prominent platform when it comes to external resources for software development [3, 27, 30, 36]. Essentially, developers use SO as a primary source to look for solutions to a programming task, or to find a practical way to fix errors. More importantly, they also report problems encountered during their daily tasks, asking for help and clarification from the community [3], leveraging the so-called *wisdom of the crowd.* Answers for questions posted on Stack Overflow can effectively help developers solve their problems. Nevertheless, even though there are many questions in Stack Overflow, only a small fraction of them get answered [48, 50]. This happens because the problems either need to be better described or appear unattractive to other developers [21]. Among others, the title does have a role to play in the visibility of posts; existing studies [6, 35, 37] attributed the low quality of posts to the informativeness of their titles. In fact, writing a concise and meaningful title is a challenging task, as it requires time and effort, as well as a thorough understanding of the whole post. In this respect, an automatic way to generate a title from the content of a post is highly desired.

Various studies have addressed the issue of Stack Overflow post title generation [9, 21, 46, 47]. `Code2Que` [9] is among the first approaches, utilizing LSTM neural networks to produce titles from source code. Though it achieves commendable prediction accuracy, `Code2Que` has certain drawbacks. Specifically, it generates titles based solely on the code snippet, overlooking the accompanying post description. This might lead to an inability to grasp the complete context of a code snippet, causing ambiguities where the same code snippet could be associated with different titles [21, 46]. To this end, `SOTitle` [21] has been conceived to transcend such limitations. Being built on top of the Transformer structure, `SOTitle` captures long-term dependencies through a multi-head attention mechanism. More importantly, `SOTitle` is also a multitask learning tool, i.e., training with posts from different programming languages. Following the same line of reasoning, various studies [9, 46, 47] also deal with multitask learning, exploiting various pre-trained language models, and obtaining an encouraging accuracy.

However, as we show later on in Section 2.2, there are two main difficulties in post title recommendation: *(i)* The potential for exposure bias, which mainly emerges from differences in how the model is trained compared to how it operates during inference; and *(ii)* The inherent randomness in sequence generation models, leading to significant fluctuations in the quality of generated titles. Altogether, this poses challenges in providing relevant recommendations.

This paper presents a practical approach to generating titles for Stack Overflow posts. We conceptualize a tool named `FILLER` to overcome the aforementioned obstacles in SO post title generation, leveraging **FI**ne-tune **L**anguage mode**L** with self improv**E**ment and post **R**anking. An empirical evaluation using real-world datasets that cover posts on 4 different programming languages demonstrated that `FILLER` is highly effective in providing recommendations. We compared `FILLER` with 5 state-of-the-art approaches for post title recommendation, including `Code2Que` [9], `SOTitle` [21], `CCBERT` [47], `M3NSCT5` [46], and `GPT3.5-turbo` [26]. More importantly, we evaluate `FILLER` against `SOTitle` and `GPT3.5-turbo` by means of a user study. The results

ESEM 2024, 20–25 October, 2024, Barcelona, Spain
2024. ACM ISBN 978-x-xxxx-xxxx-x/YY/MM...$15.00
https://doi.org/XXXXXXX.XXXXXXX

of the experiment demonstrate that FILLER obtains a promising performance, outperforming all the competitors. In this respect, our paper makes the following contributions:

▷ *Investigation.* To the best of our knowledge, this work is the first one ever that has examined the applicability of fine-tuning language model with self improvement and post ranking in generating SO titles.

▷ *Solution.* We develop FILLER, a novel tool for automatically generating titles for Stack Overflow posts using their content with code and text as input, attempting to create concise and informative titles for the posts under consideration.

▷ *Evaluation.* An empirical evaluation together with a user study has been conducted using real-world datasets to study the performance of our proposed approach, and compare it with five state-of-the-art baselines, i.e., Code2Que [9], SOTitle [21], CCBERT [47] and M3NSCT5 [46], and GPT3.5-turbo [26].

▷ *Open Science.* To facilitate future research, we publish a replication package including the dataset and source code of FILLER [2].

Structure. Section 2 reviews the related work and introduces the research motivation. We describe in detail the proposed approach in Section 3. Afterward, Section 4 presents the materials and methods used to conduct an empirical evaluation on FILLER. Section 5 reports and analyzes the experimental results, it also highlights the threats to validity of our findings. Finally, Section 6 sketches future work and concludes the paper.

2 BACKGROUND AND MOTIVATION

This section reviews state-of-the-art research dealing with the problem of generating SO posts. Based on this, we also identify the existing challenges and develop proposals to overcome them.

2.1 Related Work

Mondal et al. [25] highlighted that an increasing number of open questions on SO remain unanswered, partially attributed to the suboptimal quality of the question expression. Numerous research endeavors have been dedicated to addressing this issue, with a specific focus on crafting high-quality question titles [9, 20, 21, 46, 47, 49]. These investigations predominantly revolved around generating question titles by analyzing the content of SO posts, including problem descriptions and code snippets.

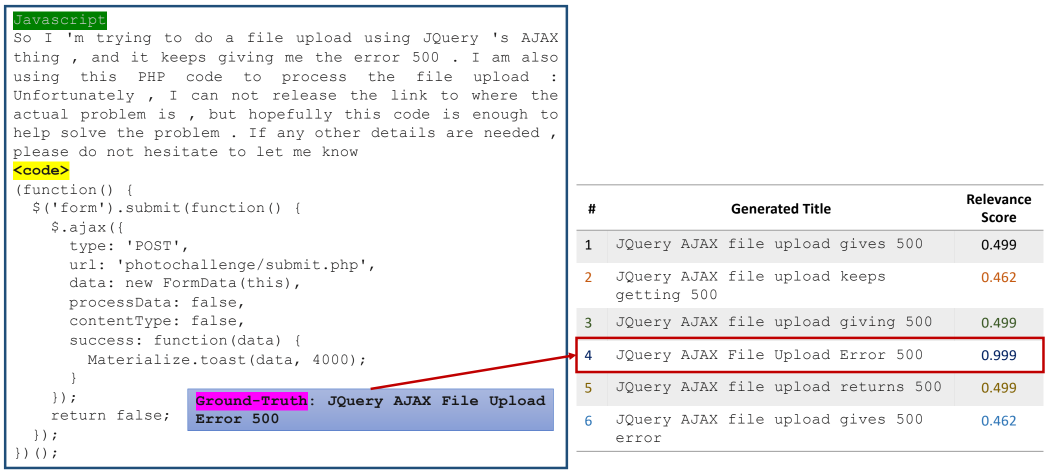
Initial studies in this domain depended solely on code snippets for generating question titles, aligning closely with the code summarization task [4, 32, 33, 48]. Gao et al. [9] introduced Code2Que to generate SO titles. This model employs bidirectional LSTMs, coupled with an attention mechanism, to encode the sequence of code tokens into hidden states and subsequently decode them into concise question titles in natural language. This method also incorporated two conventional Natural Language Processing (NLP) techniques. It implemented a coverage mechanism [38] to mitigate word repetition issues and a copy mechanism [10] to address out-of-vocabulary challenges. Though Code2Que exhibits promising performance, LSTMs may struggle to handle long-range dependencies, leading to challenges in capturing the semantic meaning of the entire source code. The most recent research efforts seek to harness the extensive dataset for pre-training language models and then fine-tuning for a specific task, resulting in state-of-the-art achievements, particularly in code summarization [1, 12] and SO post title generation [46]. Ahmad et al. [1] introduced a Transformer model, PLBART, to tasks related to programming languages such as code summarization, code generation and code translation. Subsequent studies focus on introducing multilingual representation models tailored for programming languages, as exemplified by UnixCoder [12], CodeT5 [42]. Zhang et al. [46] introduced M3NSCT5, fine-tuning the CodeT5 model to translate code sequences into relevant titles for SO posts. However, the lack of context around code snippets can lead to the same code snippets being associated with different titles, introducing ambiguity. In addressing this, M3NSCT5 emphasizes diversifying generated texts using Nucleus Sampling strategy [13] and Maximal Marginal Ranking [45], providing users with a selection of titles for a given code snippet. While our approach shares the aspect of producing multiple candidates during the inference stage, our primary goal differs–we aim to identify the most relevant title aligned with the input question description and code snippets.

Recent studies in the realm of code summarization also proved that the inclusion of additional information, such as API documentation [14], source code comments [43], alongside the source code, has demonstrated improved performance compared to relying solely on the source code. Pre-trained models with the ability of multi-modal input modelling are typically adopted in these studies. Zhang et al. [47] introduced CCBERT, which leverages the pre-trained CodeBERT model [8] to parse bi-modal contents (i.e., problem description and code snippets), incorporated with the copy mechanism [10] to handle rare tokens. Liu et al. [21] also incorporate multi-modal modeling in their approach, presenting SOTitle for generating Stack Overflow post titles.

SOTitle was constructed upon the pre-trained T5 model [28], aiming to establish a unified model for concurrent training across multiple programming languages. Subsequent research endeavors have adopted this input representation approach for Stack Overflow post title generation [20, 46, 49]. Instead of creating new titles for SO posts, Liu et al. [20] focused on reformulating question titles through the analysis of modification logs on the Stack Overflow platform. The proposed QETRA model also employs the pre-trained T5 as the backbone model to learn the title reformulation patterns from the extracted title modification logs. In contrast to previous research, QETRA employs both the question body (comprising code snippets and problem description) and the question titles provided by users to suggest candidate reformulated titles. Another task closely related to the generation of Stack Overflow post titles is the completion of question titles. Zhou et al. [48] introduced QTC4SO, an innovative approach for generating Stack Overflow post titles from bi-modal post contents and incomplete titles. Utilizing the pre-trained T5 model, QTC4SO learns potential title completion patterns and incorporates a multi-task learning strategy to leverage meaningful information from various programming languages.

2.2 Challenges

When utilizing pre-trained models (PTMs) for natural language generation tasks in general and specifically for creating SO post



```
Javascript
So I 'm trying to do a file upload using JQuery 's AJAX
thing , and it keeps giving me the error 500 . I am also
using this PHP code to process the file upload :
Unfortunately , I can not release the link to where the
actual problem is , but hopefully this code is enough to
help solve the problem . If any other details are needed ,
please do not hesitate to let me know
<code>
(function() {
  $('form').submit(function() {
    $.ajax({
      type: 'POST',
      url: 'photochallenge/submit.php',
      data: new FormData(this),
      processData: false,
      contentType: false,
      success: function(data) {
        Materialize.toast(data, 4000);
      }
    });
    return false;
  });
})();
```

Ground-Truth: JQuery AJAX File Upload Error 500

#	Generated Title	Relevance Score
1	JQuery AJAX file upload gives 500	0.499
2	JQuery AJAX file upload keeps getting 500	0.462
3	JQuery AJAX file upload giving 500	0.499
4	JQuery AJAX File Upload Error 500	0.999
5	JQuery AJAX file upload returns 500	0.499
6	JQuery AJAX file upload gives 500 error	0.462

Figure 1: Titles generated by CodeT5: The firstly ranked title is not the most relevant one.

titles, two crucial issues need attention to enhance the quality of the generated text as follows. The first issue (I1) concerns the potential for exposure bias, which mainly stems from differences in how the model is trained compared to how it operates during inference. During training, the model is designed to predict the next word in a sequence using the correct prior words from the training data, ensuring that it always operates within the correct contextual framework up to that point. However, during inference, the model generates words sequentially, relying on its own prior predictions to inform subsequent ones. As a result, any errors in early predictions can be compounded, leading to outputs that lack coherence. Enhancing the training dataset with examples that mirror the conditions experienced during inference can aid the model in learning to manage its own mistakes.

The second issue *(I2)* pertains to the inherent randomness in sequence generation models, leading to significant fluctuations in the quality of generated titles. Figure 1 depicts a motivating example, where CodeT5 was used to generate a title for a specific post,[1] which is about asking for a solution to a JavaScript snippet. The post already had an existing title: *"JQuery AJAX File Upload Error 500,"* serving as the ground-truth data. We generated 30 title candidates using CodeT5 and computed their relevance scores using the TextRank algorithm [24]. Remarkably, the results of our experiment showed that the first title produced by CodeT5 was not the most relevant one, with a relevance score of only 0.499. However, the 4th sample on the list perfectly matches the ground-truth data, earning a TextRank relevance score of 0.999. This example highlights the importance of generating multiple candidates to increase the likelihood of achieving high-quality titles. This method has been successfully implemented in several studies, e.g., those conducted to improve the performance of fault-aware neural networks [15] and neural language to code translators [32].

To overcome the aforementioned obstacles in post title generation, we develop `FILLER` on top of fine-tune language model with self improvement and post ranking. To tackle *I1*, we augment the training dataset by integrating the model's own predictions. Employing predictions as inputs during training may reduce the discrepancy between training and inference, enhancing the model's adaptability to real-world data. For *I2*, we employ a post-ranking method that produces a variety of candidates, subsequently selecting the most suitable one, thus increasing the likelihood of generating more relevant titles.

[1]The post is found in the following link: https://stackoverflow.com/questions/30010684/jquery-ajax-file-upload-error-500

3 PROPOSED SOLUTION

This section describes our proposed approach to Stack Overflow post title generation–`FILLER`–whose overall workflow is depicted in Figure 2. There are three main stages as follows: *fine-tuning*, *self-improvement* and *inference*. During the fine-tuning stage, each SO post, comprising a question description and code snippet, is combined to form a bi-modal input. Our approach adopts a multi-task learning strategy, similar to previous studies [20, 21, 46], training the model simultaneously on various programming languages. We utilize CodeT5 [42] as the backbone model and proceed with fine-tuning it on a Stack Overflow post dataset.

In addition to the conventional pre-training and *fine-tuning* paradigm, our goal is to boost the fine-tuned model's performance by augmenting the training dataset. This enhanced dataset will be employed in a further fine-tuning stage. This process is referred to as the *self-improvement* technique [34]. In the final *inference* stage, the model generates multiple candidate samples rather than producing just a single sample. Our proposed post-ranking method is designed to identify and select the highest quality title from these.

3.1 Fine-tuning PTM with multi-modal inputs

Fine-tuning using a PTM for downstream tasks is a prevalent paradigm, extensively employed in numerous SE tasks [5, 16, 22]. `FILLER` also follows this paradigm, utilizing the pre-trained CodeT5 [42] model as the backbone, then further updating its trainable parameters using a task-specific dataset consisting of pairs of post title and body.

Getting inspiration from previous research [21, 46, 47], we create a bi-modal input for each SO post by merging the code snippet with

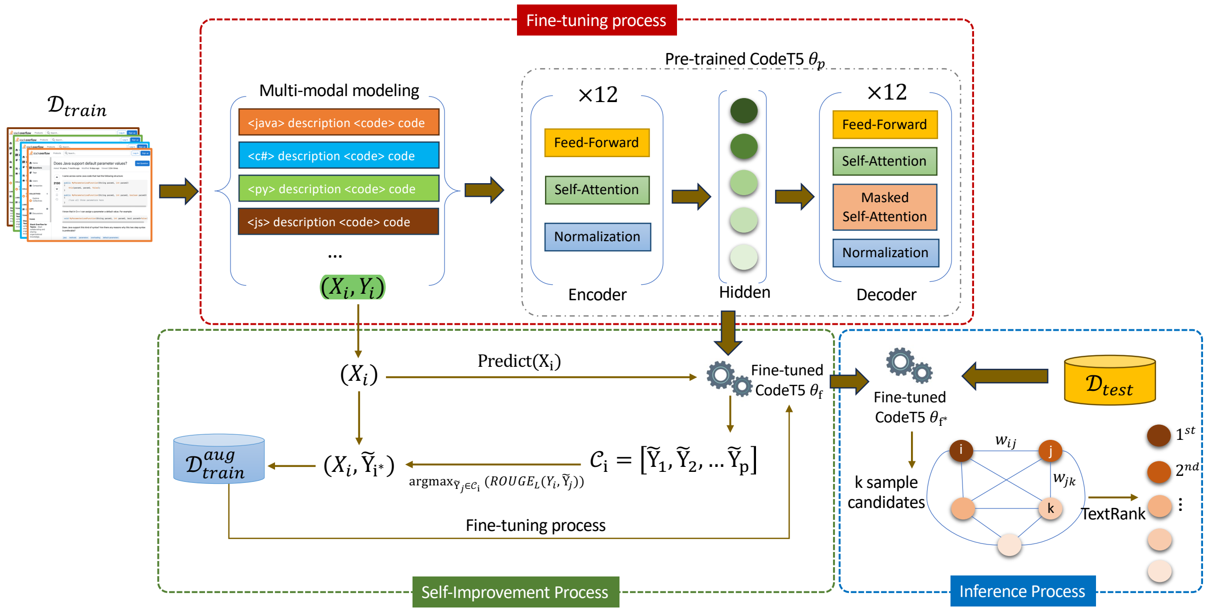

Fine-tuning process
$\mathcal{D}_{train}$
Multi-modal modeling
<java> description <code> code
<c#> description <code> code
<py> description <code> code
<js> description <code> code
...
(X_i, Y_i)
Pre-trained CodeT5 θ_p
×12
Feed-Forward
Self-Attention
Normalization
Encoder
Hidden
Masked Self-Attention
Decoder
Predict(X_i)
Fine-tuned CodeT5 θ_f
(X_i)
$\mathcal{D}^{aug}_{train}$
$(X_i, \tilde{Y}_{i^*})$
$\mathcal{C}_i = [\tilde{Y}_1, \tilde{Y}_2, ... \tilde{Y}_p]$
$\text{argmax}_{\tilde{Y}_j \in \mathcal{C}_i}(ROUGE_L(Y_i, \tilde{Y}_j))$
Fine-tuning process
Self-Improvement Process
$\mathcal{D}_{test}$
Fine-tuned CodeT5 θ_{f^*}
k sample candidates
w_{ij}
w_{jk}
TextRank
1st
2nd
Inference Process

Figure 2: The architecture of FILLER.

the question description. This combined representation is then tokenized using the Byte-Pair Encoding (BPE) method [31], which was integrated into CodeT5. This tokenizer is specifically designed to tackle the Out-of-Vocabulary (OoV) issues efficiently. We also explore a multi-task learning setting that enables training a shared model on multiple programming languages (PLs) simultaneously. This approach allows for the reuse of model weights across various PLs, thereby reducing computational costs and enhancing the model's generalization capability [19]. We prepend a prefix to the input sequence to inform the model about the specific language it is dealing with. Given a post, denoted as $\mathcal{X}$ which includes a sequence of code (X_{code}) and a sequence of question description (X_{desc}), the multi-modal input of $\mathcal{X}$ is formulated as shown in Equation 1.

$$\mathcal{X} = < prefix > \oplus X_{desc} \oplus < code > \oplus X_{code} \quad (1)$$

where $< prefix >$ denotes the programming language–specific prefix. For example, the prefix JS indicates the JavaScript programming language. The special separator $< code >$ is used to distinguish between X_{desc} and X_{code}.

We fine-tune CodeT5 [42] using the formatted SO posts. CodeT5 is built on top of the Encoder-Decoder architecture [39], with 12 blocks of feed-forward neural networks, self attention, and normalization by each size (i.e., ×12). The goal of this stage is to identify a set of parameters θ that minimizes the negative log-likelihood of the target title tokens $Y = \{y_t\}$ when conditioned on the corresponding input sequence $\mathcal{X}$ from the training dataset, given below.

$$\mathcal{L}_\theta = -\sum_{t=1}^{|Y|} \log P_\theta(y_t | y_{<t}; \mathcal{X}) \quad (2)$$

The optimal parameter θ_f is determined in Equation 3.

$$\theta_f = \arg\min_\theta \frac{1}{N} \sum_{\ell=1}^{N} \mathcal{L}^\ell_\theta \quad (3)$$

where $\mathcal{L}^\ell$ denotes the loss function for the corresponding programming language ℓ, N is the number of the programming languages. In this study, we examine Stack Overflow posts from four programming languages including Java, C#, Python and JavaScript. During the inference phase, we use the fine-tuned model in conjunction with the auto-regressive decoding method to get the predicted title $\tilde{Y}$ token by token.

3.2 Self Improvement

Pre-trained models have shown considerable versatility in handling diverse tasks, greatly diminishing the need for extensive engineering by enabling multi-modal modeling [41], However, these models typically encounter a potential exposure bias due to the misalignment between the training conditions and those during inference [11, 34, 40]. To alleviate such a bias, incorporating the model's own predictions during training can be beneficial [40]. This strategy allows the model to better manage its errors, leading to enhanced performance. Inspired by this approach, we enhance our training dataset by incorporating predictions from the initially fine-tuned model. This augmented dataset is then employed to further refine the model through an additional fine-tuning stage.

The self-improvement strategy is illustrated in Algorithm 1. Its input includes the initial training dataset $\mathcal{D}_t$, the number of generated candidates k and the initially fine-tuned CodeT5 model ($\mathcal{M}_{\theta_f}$) which is characterized by the set of parameters θ_f. This procedure yields an enhanced version of the training dataset, denoted as $\mathcal{D}^{aug}_t$, which is then employed to fine-tune the model further. For each pair consisting of a post and its ground truth title, k title candidates are generated using the initially fine-tuned CodeT5 model. The candidate that achieves the highest ROUGE–L score when compared to the ground truth title is selected for inclusion in the augmented training dataset (Lines 2–9).

Algorithm 1: Pseudo code of the Self–Improvement Process

```
Input :
    • The fine–tuned model $\mathcal{M}_{\theta_f}$
    • The original training dataset $\mathcal{D}_t$
    • $k$ that denotes the number of generated candidates
Output:
    • The augmented training dataset $\mathcal{D}_t^{aug}$
    • The new fine-tuned model $\mathcal{M}_{\theta_{f^*}}$
1  $\mathcal{D}_t^{aug} \leftarrow \varnothing$
2  foreach $(\mathcal{X}_i, Y_i) \in \mathcal{D}_t$ do
3      $\mathcal{C}_i \leftarrow \varnothing$
4      foreach $j \in [1..k]$ do
5          $\tilde{Y}_j \leftarrow \mathcal{M}_{\theta_f}.\text{predict}(\mathcal{X}_i)$
6          $\mathcal{C}_i \leftarrow \mathcal{C}_i \cup \tilde{Y}_j$
7      end
8      $\tilde{Y}_i^* \leftarrow \arg\max_{\tilde{Y}_j \in \mathcal{C}_i} ROUGE_L(Y_i, \tilde{Y}_j)$
9      $\mathcal{D}_t^{aug} \leftarrow \mathcal{D}_t^{aug} \cup (\mathcal{X}_i, \tilde{Y}_i^*)$
10 end
11 $\mathcal{M}_{\theta_{f^*}} \leftarrow \text{Fine-Tuning}(\mathcal{M}_{\theta_f}, \mathcal{D}_t^{aug})$
12 return $\mathcal{D}_t^{aug}$ and $\mathcal{M}_{\theta_{f^*}}$
```

3.3 Post Ranking

Blending ground truth with model predictions during the training phase is essential for better alignment with the inference process. Yet, in the text generation process, sequence generation models often utilize decoding strategies like beam search, greedy search, and nucleus sampling to select the best sequence of tokens [13, 46]. These methods' inherent randomness can lead to significant variability in the quality of generated text. A widely used solution to this problem involves generating a range of sample candidates and then selecting the highest quality title among them [15, 32].

In this research, our goal is to generate high-quality titles for Stack Overflow posts, with a particular emphasis on addressing the issue of high variability in generation quality by implementing a post-ranking method during the inference phase. For each Stack Overflow post input $\mathcal{X}$, the model generates a set of $\mathbb{K}$ title candidates, denoted as $\mathcal{T} = \{T_1, T_2, \ldots, T_{\mathbb{K}}\}$. To assess the relevance of these generated titles, we apply the TextRank algorithm, an unsupervised, graph-based ranking technique. In this approach, we construct a graph $\mathcal{G} = \{\mathcal{T}, \mathcal{E}\}$, where each title candidate $T_i \in \mathcal{T}$ is a node in the graph $\mathcal{G}$. We initialize an edge between every pair of nodes, with the weight of each edge determined by the cosine similarity between the two corresponding title candidates. To calculate the cosine similarity between two title candidates, we utilize the Term Frequency and Inverse Document Frequency (TF–IDF) method. We construct a token vocabulary $\mathcal{V}$ from the set of titles $\mathcal{T}$. Each title T_i is represented by a vector $v_{T_i} = w_1^i, w_2^i, \ldots, w_m^i$, where m is the size of the vocabulary, and w_j^i is calculated as follows:

$$w_j^i = tf_j^i \times idf_j = (\log f_j^i + 1) \times (\log \frac{\mathbb{K}}{\mathbb{K}_i} + 1) \tag{4}$$

with f_j^i being the frequency of the j^{th} token in title T_i, $\mathbb{K}_i$ as the number of titles that contain this token, and $\mathbb{K}$ being the total number of title candidates. The lexical similarity between the TF–IDF vectors of T_i and T_j is computed, as detailed in Equation 5.

$$Sim(T_i, T_j) = \ell_{ij} = \frac{\overrightarrow{v_{T_i}} \cdot \overrightarrow{v_{T_j}}}{\|\overrightarrow{v_{T_i}}\| \times \|\overrightarrow{v_{T_j}}\|} \tag{5}$$

where v_{T_i} and v_{T_j} denote the TF–IDF vectors of T_i, T_j, respectively. The TextRank algorithm assigns a relevance score to each node, which is iteratively refined until it converges, as in Equation 6.

$$\mathcal{S}_i = (1 - a) + a \sum_{j \neq i} \frac{\ell_{ij}}{\sum_{k \neq j} \ell_{jk}} \mathcal{S}_j \tag{6}$$

where a represents a damping factor used in the TextRank algorithm, typically set to 0.23. $\mathcal{S}_i$ and $\mathcal{S}_j$ denote the relevance scores of the i^{th} and j^{th} nodes, respectively. The title that achieves the maximum relevance score is then selected as the best one.

4 EMPIRICAL EVALUATION

In this section, we present the empirical evaluation conducted to study the performance of our proposed approach.

4.1 Research Questions

A series of experiments was performed to answer the following research questions:

RQ$_1$: *How do Pre-trained Models influence the performance of Stack Overflow post title generation?* This research question seeks to determine the efficacy of Pre-trained Models (PTMs) in generating titles for SO posts. We evaluate five distinct PTMs, namely BART [17], T5 [28], CodeT5 [42], CodeBERT [8] and UnixCoder [12]. The first two models are designed for natural language processing, whereas the latter three are specialized in understanding and representing programming languages. We applied the fine-tuning process to such pre-trained models utilizing the benchmark datasets described in Section 4.2, and by following the configuration settings outlined by Liu et al. [21].

RQ$_2$: *How effective is* `FILLER` *compared to state-of-the-art baselines in generating Stack Overflow post titles?* In this question, we compare the performance of our proposed approach to the baseline works. We choose four recent state-of-the-art models for Stack Overflow post title generation including `Code2Que` [9], `SOTitle` [21], `CCBERT` [47] and `M3NSCT5` [46], which have obtained a promising recommendation performance.

▷ `Code2Que`: Gao et al. [9] proposed the `Code2Que` model to generate Stack Overflow titles from code snippets. This model is underpinned by an LSTM encoder-decoder architecture, augmented with the integration of copy and coverage mechanisms.

▷ `SOTitle`: Liu et al. [21] fine-tuned the pre-trained T5 model on their collected Stack Overflow dataset. They employed a multi-task learning architecture in which the description and code snippet of each SO post were concatenated together for each programming language but followed a simultaneous training process.

▷ `CCBERT`: The model was proposed by Zhang et al. [47], employing CodeBERT to encode both the code snippets and question descriptions. It is further enhanced by integrating a copy attention layer to refine the generated output's quality.

▷ M3NSCT5: This model was developed to generate multiple post titles from the given code snippets [46]. The pre-trained CodeT5 model was utilized as the backbone, combined with the maximal marginal ranking strategy to select the most relevant and diverse titles from multiple generated samples.

▷ GPT3.5-turbo: To the best of our knowledge, no-one has ever applied a large language model (LLM) to post title generation. In this work, we examine how well GPT3.5-turbo [26]–an LLM–can generate suitable titles. To this end, we used a paid account from ChatGPT, and ran the experiments with its API, querying the service using tokens.

It is worth mentioning that all the baselines mentioned earlier–except Code2Que–are predominantly fine-tuned pre-trained language models for generating post titles. Notably, SOTitle and CCBERT are bi-modal models, whereas Code2Que and M3NSCT5 rely solely on code snippets. For a fair comparison with FILLER, we fine-tuned these models using code snippets and question descriptions, following the methodology established by Liu et al [21].

RQ$_3$: *Which components of* FILLER *contribute to its effectiveness?* In our work, we implemented two strategies to enhance the Stack Overflow post title generation model: a Self-Improvement approach during training and a Post-Ranking method for selecting the most pertinent title for a given question body. By means of an ablation study, this research question assesses how each of these components contributes to the overall efficacy of the proposed model.

RQ$_4$: *How are the titles generated by* FILLER *perceived by developers compared to those obtained with* SOTitle *and* GPT3.5-turbo*?* To address the limitations of automatic metrics such as ROUGE [23], which only assess the lexical overlap between the reference and generated titles, we ran a human study to evaluate the performance of FILLER compared to SOTitle and GPT3.5-turbo. We chose them due to the following reasons: *(i)* It is impractical to compare FILLER with all the baselines, as the number of samples to be examined is large; thus *(ii)* SOTitle was selected as it is among the well-established approaches, with source code implementation, allowing us to tailor the experiments; and *(iii)* GPT3.5-turbo is an LLM, having the potential to recommend relevant post titles.

Following existing work [47], this research primarily assesses two key aspects of the generated titles: *(i) Readability* and *(ii) Relevance*, with each aspect being rated on a scale from 1 to 4 (see Table 1). The former evaluates the grammatical correctness and fluency of a generated title, and this is done just by reading the title. Meanwhile, the latter measures how relevant the generated title is with respect to the actual content of the original post description as well as the ground-truth title.

We randomly selected 200 samples from the testing dataset, obtaining 600 titles generated by the three models. For the evaluation, we involved three master students in Computer Science–who are not co-authors of this study–in manually assessing the titles. The students are familiar with the Stack Overflow platform, and have experience with programming in different languages. Aiming for a fair comparison, we followed existing guidelines in conducting a user study [7]. In particular, we anonymized the origin of the titles, so as to conceal the actual tool that generates them. Each student was assigned 200 questions, and they evaluated both *Readability* and *Relevance* by reading and comparing the generated titles with the ground-truth posts. Once the students finished with the process, two senior developers–who are co-authors of this paper–were then asked to validate the results, re-evaluate, and discuss with the students to resolve any possible inconsistencies, finally reaching a consensus. It is worth noting that all the discussion was done also on anonymized post titles, aiming to avoid any bias or prejudice against any specific tools.

Table 1: Designed human evaluation criteria [47].

Criteria	Description	Score
Readability	*We evaluated if the generated title:*	
	- contains numerous grammatical errors, making it difficult to read and understand	1
	- contains a few grammatical errors but remains readable and understandable	2
	- is easy to follow and read with minimal grammatical errors	3
	- is exceptionally well-written, clear, and appealing in terms of grammatical accuracy	4
Relevance	*The generated title in relation with the original post:*	
	- It misses completely the essence of the post's content	1
	- It partially reflects the main points of the post	2
	- It aligns well with the key points of the post	3
	- It summaries perfectly the content of the post	4

4.2 Dataset

In this research, we use a benchmark dataset from a previous study [21], comprising Stack Overflow posts in four different languages: Java, C#, Python, and JavaScript (JS), which are among the most popular programming languages. Each post includes a brief title, a descriptive question, and a code snippet. The dataset encompasses 284,298 posts across these four languages, divided into training, validation, and testing subsets. For each language, there are 60,000 and 5,000 posts for training and testing, respectively. Detailed statistics of the benchmark dataset are shown in Table 2.

Table 2: Statistics of the benchmark dataset.

Language	Training	Validation	Testing
Java	60,000	3,959	5,000
C#	60,000	6,817	5,000
Python	60,000	7,742	5,000
JavaScript	60,000	5,780	5000
Total	240,000	24,298	20,000

4.3 Evaluation Metrics

To evaluate the accuracy of the generated Stack Overflow post titles, we utilize ROUGE metrics [18]. Specifically, we calculate ROUGE-1 (R-1), ROUGE-2 (R-2), and ROUGE-L (R-L), which have been extensively used in prior studies for tasks involving text generation and summarization [9, 21, 46, 47]. ROUGE–1 and ROUGE–2 rely on 1–grams and 2–grams, respectively, whereas ROUGE–L focuses on the longest continuous sequence. The recall, precision, and F1-Score for the ROUGE-k metrics are defined below.

$$\text{Recall}_{rouge-k} = \frac{\#\text{overlapped\_}k\text{\_grams}}{\#k\text{\_grams} \in \text{gold summary}}$$

$$\text{Precision}_{rouge-k} = \frac{\#overlapped\_k\_grams}{\#k\_grams \in \text{generated summary}}$$

$$\text{F1-Score}_{rouge-k} = 2 \times \frac{\text{Recall}_{rouge-k} \times \text{Precision}_{rouge-k}}{\text{Recall}_{rouge-k} + \text{Precision}_{rouge-k}}$$

where the *gold summary* refers to the original title of the SO post while the *generated summary* denotes the title generated by the underlying model. The *overlapped_k_grams* indicates the *k*_grams which appears in both the gold summary and the generated one. The recall metric calculates the proportion of *k*-grams in the reference summary that are included in the generated summary, whereas precision determines the proportion of *k*-grams in the generated summary that are present in the gold summary. The F1-Score represents a balance between recall and precision. This study adopts the F1-Score as the primary evaluation metric, in line with prior research [46, 47]. Additionally, we employ the recall metric to compare how different models perform in aligning the generated summaries with the reference ones in terms of similarity [21].

4.4 Implementation Details

The implementation of FILLER utilizes Transformers,[2] based on the checkpoint of the CodeT5–base model, which features 12 layers for both encoder and decoder, each with a hidden size of 768. The input sequences are tokenized using the Byte-Pair Encoding algorithm, characterized by a vocabulary size of 32,100. Essentially, memory consumption increases quadractically (n^2) with respect to the input length. Thus, these sequences are either truncated or padded to a maximum length of 512, so as to optimize the memory. The batching is carried out with a size of 4. We employ the AdamW optimizer, with a default learning rate of $5e^{-5}$ for both the initial fine-tuning and the self–improvement stages. The model training lasts 8 epochs during the fine-tuning stage and 4 epochs in the self–improvement phase. We set the number of title candidates to 20 and 30 for the self–improvement and post–ranking processes, respectively.

For other PTMs including BART [17], T5 [28], CodeBERT [8] and UnixCoder [12], the parameters are initialized from `bart-base`, `t5-base`, `codebert-base` and `unixcoder-base-nine`. All of the above models can be found on Hugging Face.[3] All of our experiments have been carried out on PyTorch on a single GPU RTX 2080 Ti with 12GB RAM. The evaluation metrics have been computed on experimental results using the rouge library.[4]

5 EMPIRICAL RESULTS

This section reports and analyzes the results obtained from the experiments to answer the research questions in Section 4.1.

5.1 RQ$_1$: *How do Pre-trained Models influence the performance of Stack Overflow post title generation?*

Table 3 shows the performance metrics of five different Pre-trained Models (PTMs) across four different programming languages, i.e., Java, C#, Python, and JavaScript. For each PTM, we utilized a consistent input modeling approach, as elaborated in Section 3.1. In our research, we handle the tokenized input sequences with a maximum length of 512 for all the PTMs. However, our settings slightly differ from the previous configurations suggested by Liu et al. [21]. Instead of combining 256 tokens from the description and code, we adopt a different strategy where we merge the description and code and then truncate the total to 512 tokens, placing greater emphasis on the description than the code. In fact, the number of 512 tokens is defined by the underlying Transformer architecture, and this might have an impact on long posts, i.e., those with a lot of code and text. Considering that Liu et al.'s research [21] was mainly concentrated on fine-tuning the T5 model, we also present the outcomes from their study, specifically noting them as $T5_{256+256}$. The two first rows in Table 3 demonstrate that $T5_{512}$ notably surpasses $T5_{256+256}$, as used by Liu et al. [21], across all programming languages in every performance metric. This implies that the problem description offers more significant information for title generation than the code snippet. We, therefore, apply this settings for all the other PTMs.

As can be seen in Table 3, the fine-tuned CodeT5 model outperforms all other models in terms of performance metrics for C#, Python and JavaScript, and it is comparable to the T5 model (labelled as $T5_{512}$ with our settings) for Java. This superior performance of CodeT5 can be attributed to its architecture, which is based on the T5 model but with a specialized emphasis on both natural and programming languages. This allows CodeT5 to more effectively understand the semantics of code snippets. In fact, on average, the CodeT5 model demonstrates enhancements of 0.8%, 1.9%, 12.8%, and 51.7% in ROUGE-L scores compared to the T5, BART, CodeBERT, and UnixCoder models, respectively.

> **Answer to RQ$_1$:** The fine-tuned CodeT5 model achieves the best performance in comparison to that of the other PTMs for Stack Overflow post title generation.

5.2 RQ$_2$: *How effective is* FILLER *compared to state-of-the-art baselines in generating Stack Overflow post titles?*

To assess the effectiveness of our proposed approach in generating Stack Overflow post titles, we compare FILLER with four recent and well-established baselines, namely Code2Que [9], CCBERT [47], SOTitle [21] and M3NSCT5 [46]. The comparison results across four programming languages are reported in Table 4. In general, FILLER significantly outperforms the other baselines in terms of all performance measures for all programming languages. Particularly, the performance comparison reveals that text-to-text baseline models such as SOTitle, M3NSCT5, and FILLER outperform Code2Que and CCBERT models. Notably, Code2Que, relying on a sequence-to-sequence architecture, proves to be the least effective for title generation. For instance, in terms of ROUGE–1, the average improvement across all programming languages, when compared to Code2Que, is 1.8% for CCBERT, 3.5% for SOTitle, 10.4% for M3NSCT5, and 19.5% for FILLER. The SOTitle model exhibits slightly better performance than CCBERT but is less effective when

[2]https://github.com/huggingface/transformers
[3]https://huggingface.co/models
[4]https://github.com/pltrdy/rouge

Table 3: Performance of Pre-trained Models (PTMs) including BART, T5, CodeBERT, CodeT5, and UnixCoder in title generation for Stack Overflow Posts.

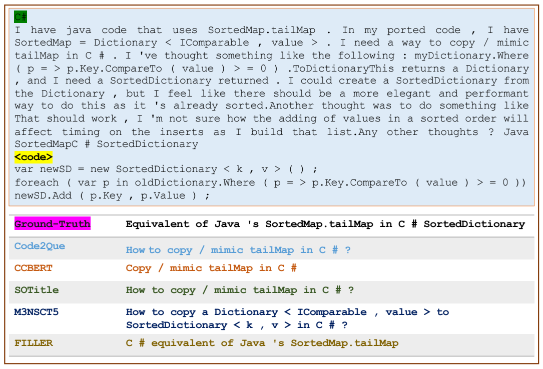
Model	Java			C#			Python			JavaScript			Average		
	R−1	R−2	R−L	R−1	R−2	R−L	R−1	R−2	R−L	R−1	R−2	R−L	R−1	R−2	R−L
T5$_{256+256}$	26.49	9.80	24.60	26.94	10.76	25.17	29.00	10.79	26.70	28.53	10.77	26.47	27.74	10.53	25.74
T5$_{512}$	**28.72**	**11.14**	**26.59**	29.03	12.07	27.21	31.39	12.37	28.87	30.96	12.65	28.80	30.03	12.06	27.87
BART	28.40	10.96	26.35	28.48	11.80	26.72	31.25	12.29	28.79	30.62	12.16	28.51	29.69	11.80	27.59
CodeBERT	25.22	9.31	23.62	25.17	10.28	23.85	28.16	10.75	26.36	27.49	10.67	25.85	26.51	10.25	24.92
UnixCoder	25.28	9.34	23.58	25.54	10.49	24.17	27.49	10.00	26.10	27.84	10.66	26.08	19.73	7.65	18.52
CodeT5	28.64	11.10	26.53	**29.24**	**12.38**	**27.31**	**31.94**	**12.71**	**29.40**	**31.23**	**12.74**	**29.16**	**30.26**	**12.23**	**28.10**

Table 4: Effectiveness of FILLER compared to state-of-the art baseline models on title generation for Stack Overflow posts.

Model	Java			C#			Python			JavaScript			Average		
	R−1	R−2	R−L	R−1	R−2	R−L	R−1	R−2	R−L	R−1	R−2	R−L	R−1	R−2	R−L
Code2Que [9]	25.25	9.17	23.57	26.55	10.57	25.01	27.93	10.36	25.87	27.50	10.40	25.70	26.81	10.13	25.04
CCBERT [47]	26.25	9.65	24.57	26.00	10.76	24.72	28.55	10.69	26.54	28.34	10.83	26.64	27.29	10.48	25.62
SOTitle [21]	26.49	9.80	24.60	26.94	10.76	25.17	29.00	10.79	26.70	28.53	10.77	26.47	27.74	10.53	25.74
M3NSCT5 [46]	27.90	10.22	25.52	28.77	11.55	26.53	31.35	11.80	28.44	30.39	11.65	27.96	29.60	11.31	27.11
GPT3.5-turbo	25.68	7.62	22.52	25.21	7.80	22.69	28.48	8.92	25.06	27.15	8.19	24.11	26.63	8.13	23.60
FILLER	**30.48**	**11.52**	**27.94**	**30.89**	**12.81**	**28.60**	**33.64**	**13.00**	**30.63**	**33.14**	**13.24**	**30.55**	**32.04**	**12.64**	**29.43**

Table 5: Ablation Study. FILLER$_{w/o\ PR}$, FILLER$_{w/o\ SI}$ and FILLER$_{w/o\ SI+PR}$ denote three variants of FILLER without Post Ranking (PR), Self Improvement (SI) and without both PR and SI, respectively.

Model	Java			C#			Python			JavaScript			Average		
	R−1	R−2	R−L	R−1	R−2	R−L	R−1	R−2	R−L	R−1	R−2	R−L	R−1	R−2	R−L
FILLER$_{w/o\ SI+PR}$	28.64	11.10	26.53	29.24	12.38	27.31	31.94	12.71	29.40	31.23	12.74	29.16	30.26	12.23	28.10
FILLER$_{w/o\ PR}$	28.93	11.26	26.89	29.27	12.46	27.45	32.17	12.78	29.64	31.77	12.89	29.59	30.54	12.35	28.39
FILLER$_{w/o\ SI}$	29.71	11.29	27.35	29.86	12.46	27.78	33.05	12.97	30.20	32.34	12.90	29.95	31.24	12.41	28.82
FILLER	**30.48**	**11.52**	**27.94**	**30.89**	**12.81**	**28.60**	**33.64**	**13.00**	**30.63**	**33.14**	**13.24**	**30.55**	**32.04**	**12.64**	**29.43**


C#

I have java code that uses SortedMap.tailMap . In my ported code , I have SortedMap = Dictionary < IComparable , value > . I need a way to copy / mimic tailMap in C # . I 've thought something like the following : myDictionary.Where (p = > p.Key.CompareTo (value) > = 0) .ToDictionaryThis returns a Dictionary , and I need a SortedDictionary returned . I could create a SortedDictionary from the Dictionary , but I feel like there should be a more elegant and performant way to do this as it 's already sorted.Another thought was to do something like That should work , I 'm not sure how the adding of values in a sorted order will affect timing on the inserts as I build that list.Any other thoughts ? Java SortedMapC # SortedDictionary

```
<code>
var newSD = new SortedDictionary < k , v > ( ) ;
foreach ( var p in oldDictionary.Where ( p = > p.Key.CompareTo ( value ) > = 0 ))
newSD.Add ( p.Key , p.Value ) ;
```

Ground-Truth	**Equivalent of Java 's SortedMap.tailMap in C # SortedDictionary**
Code2Que	How to copy / mimic tailMap in C # ?
CCBERT	Copy / mimic tailMap in C #
SOTitle	How to copy / mimic tailMap in C # ?
M3NSCT5	How to copy a Dictionary < IComparable , value > to SortedDictionary < k , v > in C # ?
FILLER	C # equivalent of Java 's SortedMap.tailMap

Figure 3: The title generated by FILLER compared to those from the baselines for a question post example.

compared to M3NSCT5 and FILLER. This observation is supported by the fact that SOTitle utilizes fine-tuned T5, which outperforms CodeBERT–the backbone model of CCBERT, in generating post titles, as indicated in the findings of the previous question.

Figure 3 shows the titles generated by FILLER and the baseline models for a question post example related to the programming language C#.[5] In this example, the titles produced by Code2Que, CCBERT, and SOTitle successfully incorporate important keywords such as "copy/mimic", "tailMap" and "C#". However, these titles lack the nuanced details of the underlying problem. In contrast, M3NSCT5 and FILLER offer a more precise expression of key information in the post. The title generated by FILLER closely aligns with the semantic meaning of the ground-truth title. While M3NSCT5 aims to enhance diversity by expanding the exploration space during token generation, often resulting in longer sentences, FILLER focuses on increasing the likelihood of producing high-quality titles by exploring a wide range of candidates and evaluating their relevance to the post. Indeed, our proposed approach outperforms M3NSCT5 across all performance metrics, achieving an average improvement of 8.2% on ROUGE−1, 11.8% on ROUGE−2, and 8.6% on ROUGE−L.

By evaluating FILLER's performance against ChatGPT with GPT3.5-turbo, as shown in Table 4, we see that FILLER significantly outperforms GPT3.5-turbo in all the metrics, including ROUGE−1, ROUGE−2, and ROUGE−L, in various languages. Specifically, the average improvements are 20.3% for ROUGE−1, 55.5% for ROUGE−2, and 24.7% for ROUGE−L.

[5]The post is available online in this link: https://bit.ly/3Hn3o2Z

Answer to RQ$_2$: `FILLER` consistently outperforms the four state-of-the-art baseline models, including `SOTitle`, `Code2Que`, `CCBERT`, `M3NSCT5` and `GPT3.5-turbo` in generating tiles for Stack Overflow posts consisting of code written in four different programming languages.

5.3 RQ$_3$: *Which components of* `FILLER` *contribute to its effectiveness?*

In this investigation, we conduct an ablation study to dissect the factors contributing to the improvement in performance. We examine three specific scenarios: (1) Excluding only the Post-Ranking component, denoted as $\texttt{FILLER}_{w/o\ PR}$; (2) Leaving out the Self-Improvement component, denoted as $\texttt{FILLER}_{w/o\ SI}$; and (3) Removing both the Self-Improvement and Post-Ranking components, denoted as $\texttt{FILLER}_{w/o\ SI+PR}$. Table 5 shows the obtained results from these experiments.

It can be observed that, without the presence of both Self-Improvement and Post-Ranking, `FILLER` experiences a consistent decline across all the metrics for various languages. For instance, in terms of ROUGE-1, $\texttt{FILLER}_{w/o\ SI+PR}$ exhibits a decrease of 6.4% for Java, 5.6% for C#, 5.3% for Python, and 6.1% for JavaScript, leading to an average reduction of 5.9%. The results are also consistent for ROUGE-2 and ROUGE-L. In the second scenario, excluding the Post-Ranking component while retaining Self-Improvement slightly boosts performance compared to the model lacking both components. Specifically, with Self Improvement active, there is an average increase of 0.93% for ROUGE-1, 0.98% for ROUGE-2, and 1.03% for ROUGE-L, compared to $\texttt{FILLER}w/o, SI + PR$. In the third scenario, including Post-Ranking but omitting Self-Improvement leads to even better performance than the second scenario. Specifically, $\texttt{FILLER}w/o, SI$ shows an average improvement of 2.3% for ROUGE-1, 0.49% for ROUGE-2, and 1.5% for ROUGE-L compared to $\texttt{FILLER}_{w/o,PR}$, highlighting the value of generating and ranking diverse titles to improve the model's output quality. In summary, our experiments indicate that the exclusion of either component from `FILLER` results in a decline in performance when generating post titles. This underscores the significance of both Self-Improvement and Post-Ranking for the effective operation of `FILLER`.

Table 6: Analysis of different $\mathbb{K}$-values in post ranking.

Top-k Ranking	Average		
	R–1	R–2	R–L
$\mathbb{K}$ = 10	31.46	12.45	29.02
$\mathbb{K}$ = 20	31.82	12.55	29.23
$\mathbb{K}$ = 30	32.04	**12.64**	29.43
$\mathbb{K}$ = 40	32.06	12.62	29.44
$\mathbb{K}$ = 50	**32.13**	12.56	**29.45**

For a more in-depth analysis of the effectiveness of Post-Ranking, we delve into the impact of selecting the $\mathbb{K}$ value during the generation of title candidates. Table 6 presents a comparative analysis of various $\mathbb{K}$-values, ranging from 10 to 50, for clarity. A list of 50 items is quite lengthy, thus representing the upper limit worth considering. The empirical data shows consistent improvements in ROUGE-1 and ROUGE-L scores across all languages when the $\mathbb{K}$-value is increased from 10 to 50. This aligns with our expectations due to the maximization-based objective of the beam search algorithm during the decoding stage, as well as the intricacies of the TextRank algorithm in the post-ranking process. The beam search algorithm prioritizes tokens with the highest probability, emphasizing key tokens from the input. Consequently, increasing the number of sample candidates enhances the ability to capture these key tokens. On the other hand, TextRank tends to assign high ranking scores to titles containing common key tokens from others, leading to the selection of longer titles with a greater number of important tokens. However, longer titles may have a detrimental effect on ROUGE-2. Table 6 shows that compared to the $\mathbb{K}$-value of 10, the $\mathbb{K}$-value of 50 yields an average improvement of 1.65% for ROUGE-1 and 1.07% for ROUGE-L. Conversely, the $\mathbb{K}$-value exceeding 30 results in a performance decline in terms of ROUGE-2. Balancing between speed and quality is crucial, making the $\mathbb{K}$-value of 30 the preferred choice, considering the impracticality of generating an excessive number of sample titles.

Answer to RQ$_3$: Without Self-Improvement and Post-Ranking, the performance of `FILLER` decreases by 5.9% and 3.4% and 4.7% in terms of ROUGE-1, ROUGE-2 and ROUGE-L, respectively. A $\mathbb{K}$ value set to 30 for the Post-Ranking process provides a balanced compromise between the speed and quality of the generated titles. All the components of `FILLER` contribute positively to its performance.

5.4 RQ$_4$: *How are the titles generated by* `FILLER` *perceived by developers compared to those obtained with* `SOTitle` *and* `GPT3.5-turbo`*?*

Before analyzing the results, we extract some evaluation examples from the user study, and show them in Table 7 to illustrate how the outcome looks like with respect to *Readability* (R$_1$) and *Relevance* (R$_2$).[6] It is evident that `GPT3.5-turbo` usually generates *verbal* titles, containing a lot of text (R$_1$ is considerably high) but not relevant to the ground-truth ones, i.e., all the samples get 1 as R$_2$. `SOTitle` is better compared to `GPT3.5-turbo` in recommending highly relevant titles. Among others, `FILLER` is able to produce more *factual* titles, being close to the original ones, i.e., having 3 and 4 as the Relevance scores. This is also the case by the remaining samples, as we show in the following analysis.

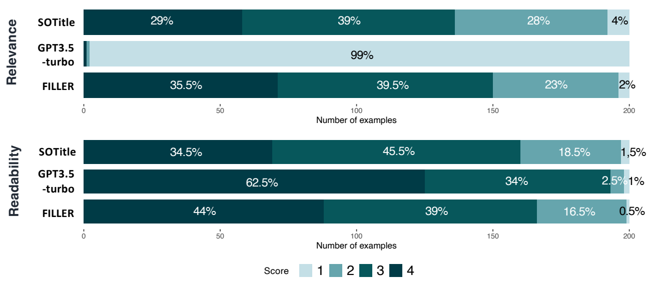


Figure 4: Evaluation results from the user study.

[6]Due to space limit, we present only 4 representative examples. Interested readers are kindly referred to the replication package [2] for more details.

Table 7: Examples of the evaluation scores from the user study.

		SOTitle			GPT3.5-turbo			FILLER		
No.	**Ground-truth title**	**Generated title**	R_1	R_2	**Generated title**	R_1	R_2	**Generated title**	R_1	R_2
1	React-highcharts scrollbar feature not working	react-highchart scrollbar not working	2	4	How to handle errors and rewrite mapping logic using Ramda in functional programming for indexing target columns in CSV header?	4	1	react-highcharts bar scrollbar not working	2	4
2	Why is 'this' undefined in the debugger but printable in a console.log?	Why does console.log set a breakpoint on console.log?	3	1	What is the purpose of the "(0, foo)()" syntax in the JavaScript code snippet provided?	3	1	Why is ‘this‘ undefined in console.log?	4	3
3	Why does an unhandled exception in this background thread not terminate my process?	Unhandled exception in foreground and background threads	4	2	How to achieve constant rotation of a character's head without flipping the character when rotated more than 180 degrees in Unity?	3	1	Why does an unhandled exception in a background thread not cause the process to terminate ?	3	4
4	"list.toArray (T [ ] a), what if the “T” is a “runtime type”?"	How to convert a list into an array if it is a runtime type”?	3	2	Java 9 issue with casting toArray in Arrays.asList causing code to not work	3	1	List.toArray (T [ ] a), what if the T is a “runtime type”?	2	4

Figure 4 shows the evaluation results on human study of SOTitle, GPT3.5-turbo, and FILLER. In terms of Relevance, the titles generated by GPT3.5-turbo significantly deviate from the content of the post and the corresponding ground-truth titles, with 99% of the examples receiving a score of 1 for the predicted titles. Essentially, FILLER performs better, achieving the top score of 4 in 35.5% of cases, which is 6.5% higher than SOTitle, showcasing that our model more precisely encapsulates the content of SO posts. Both SOTitle and FILLER effectively capture the essential points of the posts, with relevance scores of 3 and 2 in 67% and 62.5% of samples, respectively. Regarding Readability, GPT3.5-turbo demonstrates its strength with 62.5% of examples scoring the highest mark of 4, highlighting its ability to craft titles that are well-written and appealing to human readers. The titles from both FILLER and SOTitle are clear and easy to comprehend, with readability scores of 4 and 3 in 83% and 80% of the samples, respectively.

Finally, we validate the null hypothesis that there are no statistical differences between the performance of FILLER and that of SOTitle as well as of GPT3.5-turbo. Computing the Wilcoxon rank sum test [44] on the scores obtained by each pair of tools, i.e., FILLER vs. SOTitle and FILLER vs. GPT3.5-turbo, we see that p-values for R_1 and R_2 are smaller than 1.06e-05 and 2.2e-16, respectively. Considering the 95% significance level (p-value < 0.05), it is evident that the p-values are lower than 5e-02. Thus, we reject the null hypothesis and conclude that the performance improvement obtained by FILLER is statistically significant.

Answer to RQ$_4$: The user study demonstrates that FILLER is capable of generating high-quality titles, which capture well the intrinsic characters of Stack Overflow posts, resembling those given by humans. The obtained performance gain is statistically significant.

5.5 Threats to validity

We anticipate that there might be the following threats to validity of our findings.

Internal validity. This is related to the fact that our evaluation resembles real-world scenarios. We used existing datasets that have been widely used for the same purpose. The comparison with the baselines was done using their original implementation, i.e., we ran the tools provided in their replication package, attempting to mitigate any possible threats to internal validity. The user study was conducted following existing guidelines [7], first by anonymizing the posts, and then discussing among the evaluators to aim for sound evaluation, as well as to avoid any bias. A probable threat is the amount of information discarded during the text encoding phase. We truncated the input sentences to 512 tokens, as this was pre-defined by the Transformers library. This can be subject to loss of information if the posts are longer than this threshold.

External validity. This concerns the generalizability of the findings outside this study. In the evaluation, we experimented with datasets covering four different programming languages, i.e., Java, C#, Python, and JavaScript, and our findings are valid for posts containing code in these languages.

Construct validity. This threat is about the setup and measurement in the study, i.e., if they resemble real-world situations. We attempted to simulate a real scenario of generating titles for Stack Overflow posts, by means of cross validation. In particular, the dataset was split into two independent parts, including a training set and a testing set. This simulates the scenario where the items in the training data correspond to the posts available for the recommendation process. Meanwhile, an item in the testing data corresponds to the post under consideration, and the models are expected to generate a title for each post.

6 CONCLUSION AND FUTURE WORK

This paper presented an approach named FILLER to Stack Overflow post title generation. Our tool leverages fine-tuning language model with self-improvement and post-ranking. By incorporating the model's own predictions into the training process, it can learn from its errors, thus mitigating the effects of exposure bias. Moreover, a post-ranking method was employed to produce a variety of sample candidates, subsequently selecting the most suitable one, thus increasing the likelihood of generating more relevant titles. An empirical evaluation using real-world datasets with posts covering four different programming languages shows that FILLER obtains a promising recommendation performance, thus outperforming four state-of-the-art baselines for post title recommendation.

For future work, we plan to test FILLER with data from different sources and posts related to other programming languages. Moreover, we will also incorporate various post ranking mechanisms to improve the recommendation performance further.

REFERENCES

[1] Wasi 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*. 2655–2668.

[2] Anonymous Authors. 2024. Artifacts: Good things come in three: Generating Post Titles with Pre-Trained Models, Self Improvement and Post Ranking. Figshare. https://figshare.com/s/4cd6bfc77c8afdc99992

[3] Stefanie Beyer, Christian Macho, Massimiliano Di Penta, and Martin Pinzger. 2021. What Kind of Questions Do Developers Ask on Stack Overflow? A Comparison of Automated Approaches to Classify Posts Into Question Categories. In *Software Engineering 2021, Fachtagung des GI-Fachbereichs Softwaretechnik, 22.-26. Februar 2021, Braunschweig/Virtuell (LNI, Vol. P-310)*, Anne Koziolek, Ina Schaefer, and Christoph Seidl (Eds.). Gesellschaft für Informatik e.V., 27–28. https://doi.org/10.18420/SE2021_03

[4] Wuyan Cheng, Po Hu, Shaozhi Wei, and Ran Mo. 2022. Keyword-guided abstractive code summarization via incorporating structural and contextual information. *Information and Software Technology* 150 (2022), 106987.

[5] Yiu Wai Chow, Max Schäfer, and Michael Pradel. 2023. Beware of the unexpected: Bimodal taint analysis. *arXiv preprint arXiv:2301.10545* (2023).

[6] Denzil Correa and Ashish Sureka. 2013. Fit or unfit: analysis and prediction of 'closed questions' on stack overflow. In *Conference on Online Social Networks, COSN'13, Boston, MA, USA, October 7-8, 2013*, S. Muthu Muthukrishnan, Amr El Abbadi, and Balachander Krishnamurthy (Eds.). ACM, 201–212. https://doi.org/10.1145/2512938.2512954

[7] Juri Di Rocco, Davide Di Ruscio, Claudio Di Sipio, Phuong Thanh Nguyen, and Riccardo Rubei. 2021. Development of recommendation systems for software engineering: the CROSSMINER experience. *Empir. Softw. Eng.* 26, 4 (2021), 69. https://doi.org/10.1007/S10664-021-09963-7

[8] Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, et al. 2020. CodeBERT: A Pre-Trained Model for Programming and Natural Languages. In *Findings of the Association for Computational Linguistics: EMNLP 2020*. 1536–1547.

[9] Zhipeng Gao, Xin Xia, David Lo, John Grundy, and Yuan-Fang Li. 2021. Code2Que: A tool for improving question titles from mined code snippets in stack overflow. In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 1525–1529.

[10] Jiatao Gu, Zhengdong Lu, Hang Li, and Victor OK Li. 2016. Incorporating Copying Mechanism in Sequence-to-Sequence Learning. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 1631–1640.

[11] Nuno M Guerreiro, Duarte M Alves, Jonas Waldendorf, Barry Haddow, Alexandra Birch, Pierre Colombo, and André FT Martins. 2023. Hallucinations in large multilingual translation models. *Transactions of the Association for Computational Linguistics* 11 (2023), 1500–1517.

[12] Daya Guo, Shuai Lu, Nan Duan, Yanlin Wang, Ming Zhou, and Jian Yin. 2022. UniXcoder: Unified Cross-Modal Pre-training for Code Representation. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 7212–7225.

[13] Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. 2019. The Curious Case of Neural Text Degeneration. In *International Conference on Learning Representations*.

[14] Xing HU, Ge LI, Xin XIA, David LO, Shuai LU, and Zhi JIN. [n. d.]. Summarizing source code with transferred API knowledge.(2018). In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelli-gence (IJCAI 2018), Stockholm, Sweden, 2018 July 13*, Vol. 19. 2269–2275.

[15] Jeevana Priya Inala, Chenglong Wang, Mei Yang, Andres Codas, Mark Encarnación, Shuvendu Lahiri, Madanlal Musuvathi, and Jianfeng Gao. 2022. Fault-aware neural code rankers. *Advances in Neural Information Processing Systems* 35 (2022), 13419–13432.

[16] Thanh Le-Cong, Duc-Minh Luong, Xuan Bach D Le, David Lo, Nhat-Hoa Tran, Bui Quang-Huy, and Quyet-Thang Huynh. 2023. Invalidator: Automated patch correctness assessment via semantic and syntactic reasoning. *IEEE Transactions on Software Engineering* (2023).

[17] 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*. 7871–7880.

[18] Chin-Yew Lin. 2004. Rouge: A package for automatic evaluation of summaries. In *Text summarization branches out*. 74–81.

[19] Fang Liu, Ge Li, Yunfei Zhao, and Zhi Jin. 2020. Multi-task learning based pre-trained language model for code completion. In *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering*. 473–485.

[20] Ke Liu, Xiang Chen, Chunyang Chen, Xiaofei Xie, and Zhanqi Cui. 2023. Automated Question Title Reformulation by Mining Modification Logs From Stack Overflow. *IEEE Transactions on Software Engineering* (2023).

[21] Ke Liu, Guang Yang, Xiang Chen, and Chi Yu. 2022. SOTitle: A Transformer-based Post Title Generation Approach for Stack Overflow. In *2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE Computer Society, 577–588.

[22] Yue Liu, Thanh Le-Cong, Ratnadira Widyasari, Chakkrit Tantithamthavorn, Li Li, Xuan-Bach D Le, and David Lo. 2023. Refining ChatGPT-generated code: Characterizing and mitigating code quality issues. *arXiv preprint arXiv:2307.12596* (2023).

[23] Antonio Mastropaolo, Matteo Ciniselli, Massimiliano Di Penta, and Gabriele Bavota. 2024. Evaluating Code Summarization Techniques: A New Metric and an Empirical Characterization. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering* (<conf-loc>, <city>Lisbon</city>, <country>Portugal</country>, </conf-loc>) *(ICSE '24)*. Association for Computing Machinery, New York, NY, USA, Article 218, 13 pages. https://doi.org/10.1145/3597503.3639174

[24] Rada Mihalcea and Paul Tarau. 2004. TextRank: Bringing Order into Text. In *Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing , EMNLP 2004, A meeting of SIGDAT, a Special Interest Group of the ACL, held in conjunction with ACL 2004, 25-26 July 2004, Barcelona, Spain*. ACL, 404–411. https://aclanthology.org/W04-3252/

[25] Saikat Mondal, CM Khaled Saifullah, Avijit Bhattacharjee, Mohammad Masudur Rahman, and Chanchal K Roy. 2021. Early detection and guidelines to improve unanswered questions on stack overflow. In *14th Innovations in software engineering conference (formerly known as India software engineering conference)*. 1–11.

[26] OpenAI. 2023. GPT-3.5-Turbo. https://www.openai.com. Accessed: [15 Mar 2024].

[27] Luca Ponzanelli, Gabriele Bavota, Massimiliano Di Penta, Rocco Oliveto, and Michele Lanza. 2014. Mining StackOverflow to Turn the IDE into a Self-Confident Programming Prompter. In *Proceedings of the 11th Working Conference on Mining Software Repositories* (Hyderabad, India) *(MSR 2014)*. Association for Computing Machinery, New York, NY, USA, 102–111. https://doi.org/10.1145/2597073.2597077

[28] 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.

[29] Martin Robillard, Robert Walker, and Thomas Zimmermann. 2010. Recommendation Systems for Software Engineering. *IEEE Software* 27, 4 (2010), 80–86. https://doi.org/10.1109/MS.2009.161

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

[31] Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. Neural Machine Translation of Rare Words with Subword Units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Katrin Erk and Noah A. Smith (Eds.). Association for Computational Linguistics, Berlin, Germany, 1715–1725. https://doi.org/10.18653/v1/P16-1162

[32] Freda Shi, Daniel Fried, Marjan Ghazvininejad, Luke Zettlemoyer, and Sida I Wang. 2022. Natural Language to Code Translation with Execution. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*. 3533–3546.

[33] Ze Tang, Xiaoyu Shen, Chuanyi Li, Jidong Ge, Liguo Huang, Zhelin Zhu, and Bin Luo. 2022. AST-trans: Code summarization with efficient tree-structured attention. In *Proceedings of the 44th International Conference on Software Engineering*. 150–162.

[34] Hung To, Nghi Bui, Jin L.C. Guo, and Tien Nguyen. 2023. Better Language Models of Code through Self-Improvement. In *Findings of the Association for Computational Linguistics: ACL 2023*, Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki (Eds.). Association for Computational Linguistics, Toronto, Canada, 12994–13002. https://doi.org/10.18653/v1/2023.findings-acl.823

[35] László Tóth and László Vidács. 2019. Comparative Study of The Performance of Various Classifiers in Labeling Non-Functional Requirements. *Inf. Technol. Control.* 48, 3 (2019), 432–445. https://doi.org/10.5755/J01.ITC.48.3.21973

[36] Christoph Treude and Martin P. Robillard. 2016. Augmenting API Documentation with Insights from Stack Overflow. In *Proceedings of the 38th International Conference on Software Engineering* (Austin, Texas) *(ICSE '16)*. Association for Computing Machinery, New York, NY, USA, 392–403. https://doi.org/10.1145/2884781.2884800

[37] Jan Trienes and Krisztian Balog. 2019. Identifying Unclear Questions in Community Question Answering Websites. In *Advances in Information Retrieval*

- *41st European Conference on IR Research, ECIR 2019, Cologne, Germany, April 14-18, 2019, Proceedings, Part I (Lecture Notes in Computer Science, Vol. 11437)*, Leif Azzopardi, Benno Stein, Norbert Fuhr, Philipp Mayr, Claudia Hauff, and Djoerd Hiemstra (Eds.). Springer, 276–289. https://doi.org/10.1007/978-3-030-15712-8_18

[38] Zhaopeng Tu, Zhengdong Lu, Yang Liu, Xiaohua Liu, and Hang Li. 2016. Modeling Coverage for Neural Machine Translation. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 76–85.

[39] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems* 30 (2017).

[40] Chaojun Wang and Rico Sennrich. 2020. On Exposure Bias, Hallucination and Domain Shift in Neural Machine Translation. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. 3544–3552.

[41] Xiao Wang, Guangyao Chen, Guangwu Qian, Pengcheng Gao, Xiao-Yong Wei, Yaowei Wang, Yonghong Tian, and Wen Gao. 2023. Large-scale multi-modal pre-trained models: A comprehensive survey. *Machine Intelligence Research* (2023), 1–36.

[42] Yue Wang, Weishi Wang, Shafiq Joty, and Steven CH 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*. 8696–8708.

[43] Bolin Wei, Yongmin Li, Ge Li, Xin Xia, and Zhi Jin. 2020. Retrieve and refine: exemplar-based neural comment generation. In *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering*. 349–360.

[44] Frank Wilcoxon. 1992. *Individual Comparisons by Ranking Methods*. Springer New York, New York, NY, 196–202. https://doi.org/10.1007/978-1-4612-4380-9_16

[45] Tao Yang and Qingyao Ai. 2021. Maximizing marginal fairness for dynamic learning to rank. In *Proceedings of the Web Conference 2021*. 137–145.

[46] Fengji Zhang, Jin Liu, Yao Wan, Xiao Yu, Xiao Liu, and Jacky Keung. 2023. Diverse title generation for Stack Overflow posts with multiple-sampling-enhanced transformer. *Journal of Systems and Software* 200 (2023), 111672.

[47] Fengji Zhang, Xiao Yu, Jacky Keung, Fuyang Li, Zhiwen Xie, Zhen Yang, Caoyuan Ma, and Zhimin Zhang. 2022. Improving Stack Overflow question title generation with copying enhanced CodeBERT model and bi-modal information. *Information and Software Technology* 148 (2022), 106922.

[48] Yu Zhou, Juanjuan Shen, Xiaoqing Zhang, Wenhua Yang, Tingting Han, and Taolue Chen. 2022. Automatic source code summarization with graph attention networks. *Journal of Systems and Software* 188 (2022), 111257.

[49] Yanlin Zhou, Shaoyu Yang, Xiang Chen, Zichen Zhang, and Jiahua Pei. 2023. QTC4SO: Automatic Question Title Completion for Stack Overflow. In *2023 IEEE/ACM 31st International Conference on Program Comprehension (ICPC)*. IEEE, 1–12.

[50] Wenhan Zhu, Haoxiang Zhang, Ahmed E. Hassan, and Michael W. Godfrey. 2022. An empirical study of question discussions on Stack Overflow. *Empir. Softw. Eng.* 27, 6 (2022), 148. https://doi.org/10.1007/S10664-022-10180-Z