The Journal of Systems & Software 200 (2023) 111672

Contents lists available at ScienceDirect

The Journal of Systems & Software

journal homepage: www.elsevier.com/locate/jss

Diverse title generation for Stack Overflow posts with multiple-sampling-enhanced transformer☆

Fengji Zhang[a], Jin Liu[a,*], Yao Wan[b], Xiao Yu[c,d,**], Xiao Liu[e], Jacky Keung[f]

[a] School of Computer Science, Wuhan University, Wuhan, China
[b] School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan, China
[c] Sanya Science and Education Innovation Park of Wuhan University of Technology, Sanya, China
[d] School of Computer Science and Artificial Intelligence, Wuhan University of Technology, Wuhan, China
[e] School of Information Technology, Deakin University, Geelong, Australia
[f] Department of Computer Science, City University of Hong Kong, Hong Kong, China

ARTICLE INFO

Article history:
Received 24 August 2022
Received in revised form 11 January 2023
Accepted 28 February 2023
Available online 2 March 2023

Keywords:
Stack Overflow
Title generation
CodeT5
Nucleus sampling
Maximal marginal ranking

ABSTRACT

Stack Overflow is one of the most popular programming communities where developers can seek help for their encountered problems. Nevertheless, if inexperienced developers fail to describe their problems clearly, it is hard for them to attract sufficient attention and get the anticipated answers. To address such a problem, we propose M_3NSCT5, a novel approach to automatically generate multiple post titles from the given code snippets. Developers may take advantage of the generated titles to find closely related posts and complete their problem descriptions. M_3NSCT5 employs the CodeT5 backbone, which is a pre-trained Transformer model with an excellent language understanding and generation ability. To alleviate the ambiguity issue that the same code snippets could be aligned with different titles under varying contexts, we propose the maximal marginal multiple nucleus sampling strategy to generate multiple high-quality and diverse title candidates at a time for the developers to choose from. We build a large-scale dataset with 890,000 question posts covering eight programming languages to validate the effectiveness of M_3NSCT5. The automatic evaluation results on the BLEU and ROUGE metrics demonstrate the superiority of M_3NSCT5 over six state-of-the-art baseline models. Moreover, a human evaluation with trustworthy results also demonstrates the great potential of our approach for real-world applications.

1. Introduction

Stack Overflow (SO) is one of the most popular Question& Answering websites for developers to seek answers to programming problems. However, it remains a challenge (Chatterjee et al., 2020; Mondal et al., 2021; Rubei et al., 2020) to help developers write high-quality question posts that attract sufficient attention from potential experts. Especially, non-English speaking or inexperienced developers may struggle to clearly describe their encountered problems, let alone summarize the problems into informative titles. One way for developers to improve the quality of their posts is to search for related posts with the problematic code snippets. If no answers are found, this process can still help developers gain a better understanding of their problems and complete the posts. Nonetheless, previous studies (Gao et al., 2020; Zhang et al., 2022b; Liu et al., 2022a; Gao et al., 2022) demonstrated the unsatisfying performance of the commonly used retrieval methods like TF-IDF and BM25 (Robertson and Zaragoza, 2009) on searching related posts with given code snippets. First, such retrieval methods calculate the lexical overlap and ignore the essential semantic similarity. Second, different from natural language queries, code snippets usually have very long contexts and plentiful user-defined tokens, making it hard to extract lexical features.

Recently, Gao et al. (2020) proposed an end-to-end generation model to automatically produce post titles with the given code snippets. First, they train an LSTM (Long Short-Term Memory) (Hochreiter and Schmidhuber, 1997) model on a large-scale dataset collected from Stack Overflow, which contains pairs of code snippets and post titles. Then, a developer could provide the model with code snippets to get a generated post title that summarizes the problem. The generated titles are coherent and informative, which will help developers understand their problems and find related posts more easily. However, as suggested

☆ Editor: Christoph Treude.

* Corresponding author.
** Corresponding author at: Sanya Science and Education Innovation Park of Wuhan University of Technology, Sanya, China.
E-mail addresses: zhangfengji@whu.edu.cn (F. Zhang), jinliu@whu.edu.cn (J. Liu), wanyao@hust.edu.cn (Y. Wan), xiaoyu@whut.edu.cn (X. Yu), xiao.liu@deakin.edu.au (X. Liu), jacky.keung@cityu.edu.hk (J. Keung).

https://doi.org/10.1016/j.jss.2023.111672
0164-1212/


Where in my django app do I implement this get_client_ip() function?
https://stackoverflow.com/questions/57107116

I have a Custom User model that takes user ip address. **I want to add the IP address of the user upon completion of the sign up form. Where do I implement the below code?** I am not sure whether to put this into my forms.py or views.py file.

```
def get_client_ip(request):
    x_forwarded_for = request.META.get('HTTP_X_FORWARDED_FOR')
    if x_forwarded_for:
        ip = x_forwarded_for.split(',')[0]
    else:
        ip = request.META.get('REMOTE_ADDR')
    return ip
```

I expect to be able to save the user's ip address into my custom user table upon sign up.

How get the "real" user ip address?
https://stackoverflow.com/questions/13076259

i tried to get the "real" user ip address, it's working, but i'm not getting my own ip (the real one), checking the ip with google map, after convert it to long - lat, google map is showing the location of my ISP.

```
def get_client_ip(request):
    x_forwarded_for = request.META.get('HTTP_X_FORWARDED_FOR')
    if x_forwarded_for:
        ip = x_forwarded_for.split(',')[0]
    else:
        ip = request.META.get('REMOTE_ADDR')
    return ip
```

What i want to do, is display a map from the exact location from the user sent a article.

Fig. 1. Illustration of the *ambiguity* issue — posts with the same code snippets have different titles under varying contexts.

by Liu et al. (2022a), the same code snippets could be aligned with different titles under varying contexts, which is reasonable because code snippets without surrounding context can be unclear in meaning and open to interpretation. In this study, we refer to the difficulty in determine the intention behind a code snippet without sufficient context as the *ambiguity* issue. For example, in Fig. 1, the two SO posts ask different questions and have different titles. However, they have the same code snippets that implement the Python function *get_client_ip*. Liu et al. (2022a) then proposed to tackle the issue by leveraging the surrounding text descriptions in the post body to eliminate the semantic ambiguity of code snippets. Nonetheless, it remains an open challenge to generate the expected post titles when developers cannot provide precise descriptions of their problems.

To mitigate this challenge, we reformulate the title generation task as generating multiple candidate titles simultaneously under the condition that only code snippets are provided. Since code snippets can be ambiguous without the surrounding context, we could offer the developers an acceptable amount of candidate titles to choose from. But this will pose a new challenge of improving the diversity of generated titles while keeping the quality so that the titles can nicely summarize the code snippets as well as cover different intentions under varying contexts. To this end, we propose M_3NSCT5, a novel approach to generate high-quality and diverse post titles from the given code snippets. **M_3NSCT5** is a hybrid method combining the **M**aximal **M**arginal **M**ultiple **N**ucleus **S**ampling strategy and the **C**ode**T5** model. Specifically, we employ the state-of-the-art sequence-to-sequence generation model CodeT5 (Wang et al., 2021) as the backbone of M_3NSCT5 to generate titles with higher quality. CodeT5 is a Transformer-based (Vaswani et al., 2017) model pre-trained on a large-scale code-related corpus. It has strong code understanding and text generation capabilities and is able to capture long-range dependencies more effectively than traditional LSTMs (Khandelwal et al., 2018; Chen et al., 2021b; Ma et al., 2022; Yang et al., 2021a; Zhen et al., 2022). To address the issue of *ambiguity* and generate more diverse titles, we employ nucleus sampling (Holtzman et al., 2019) during decoding instead of the commonly used beam search. While nucleus sampling can produce samples with high variance in quality, we propose the maximal marginal ranking strategy to select the highest quality and diverse titles from the samples. In this way, we can tackle the *ambiguity* issue by offering multiple title candidates for developers to choose from. The following are the top three titles generated by M_3NSCT5 for the code snippet in Fig. 1:

(1) How to get the client IP address in Django
(2) Why is HTTP_X_FORWARDED_FOR used here
(3) Django - reverse proxy setup not working

As can be seen, the generated titles are coherent and able to address a range of potential intentions. It demonstrates the effectiveness of our approach in generating high-quality and diverse post titles. To comprehensively evaluate the effectiveness of our approach, we conduct an empirical study by raising the following Research Questions (RQs):

RQ-1: What is the prevalence of the *ambiguity* issue? We conduct a human study on a sample of 300 posts from our dataset, where human evaluators are asked to determine whether each post contains ambiguous code snippets. The results of the study indicate that a significant proportion of the posts hava *ambiguity* issue, highlighting the need and usefulness of our proposed approach.

RQ-2: Does our approach outperform state-of-the-art baselines under automatic evaluation? We build a large-scale dataset D_{so} with around 890,000 high-quality SO posts covering eight programming languages. We employ BLEU (Papineni et al., 2002) and ROUGE (Lin and Och, 2004) as the automatic evaluation metrics and choose six baseline models (i.e., BM25 Robertson and Zaragoza, 2009, Code2Que Gao et al., 2020, BART Lewis et al., 2020, CCBERT Zhang et al., 2022b, SOTtitle Liu et al., 2022a, and PLBART Ahmad et al., 2021) for comparison. Experimental results show that M_3NSCT5 outperforms all the baselines by a large margin, having an around 9% improvement over the second best performing PLBART on average of different experimental settings.

RQ-3: How effective is our maximal marginal multiple nucleus sampling? We compare the performance of our sampling strategy with beam search and vanilla random nucleus sampling. Results show that our method could improve both the quality and diversity of generated titles, especially when the number of output titles is limited to a small value ($\leq$5), making it suitable for real-world applications.

RQ-4: What is the performance of our approach under human evaluation? To compensate for the non-intuitive automatic evaluation, we recruit six experienced programmers to perform an additional human evaluation. Participants are required to score the titles generated by M_3NSCT5, PLBART, and BM25 involving three programming languages on the *Readability*, *Correlation*, *Diversity*, and *Usability* criteria. Results show that our approach has better performance under human-centered evaluation.

The contributions of this paper are as follows:

- We propose M_3NSCT5, a novel approach combining the pre-trained CodeT5 model and the maximal marginal multiple nucleus sampling strategy, which could improve the quality and diversity of generated SO titles.
- We collect a large-scale dataset containing 890,000 high-quality posts covering eight programming languages and demonstrate the effectiveness of our approach under automatic and human evaluation.
- We have released the source code and processed dataset[1] to facilitate future research.

[1] https://github.com/zfj1998/M3NSCT5.

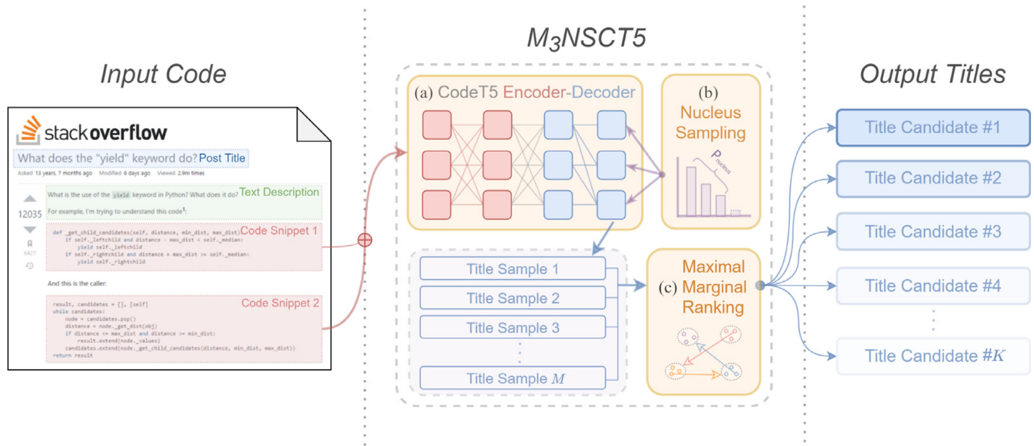


Fig. 2. The overall framework of our approach for Stack Overflow post title generation. Given the input code snippets, M_3NSCT5 can produce multiple title candidates. There are three critical components inside M_3NSCT5, namely the CodeT5 backbone, the nucleus sampling method, and the maximal marginal ranking strategy.

We organize the rest of this paper as follows: Section 2 introduces the details of our proposed approach. Section 3 describes the basic setup of our experiment, including the construction of the experimental dataset, hyper-parameter settings, baseline models, and evaluation metrics. Section 4 presents the experimental results. Section 5 discusses the limitations of our approach. Section 6 introduces the related works. Section 7 discusses threats to the validity of our work. Finally, we conclude this paper and introduce the future work in Section 8.

2. The proposed approach

Generating post titles from code snippets can be seen as a PL-to-NL (Programming Language to Natural Language) generation task. Fig. 2 illustrates the overall framework of our M_3NSCT5, a novel end-to-end approach that could improve the quality and diversity of the post titles generated from the code snippets. Specifically, we employ CodeT5 as the backbone, which takes in the code snippets and generates post titles. We further incorporate the nucleus sampling and maximal marginal ranking strategy to produce a set of high-quality and diverse title candidates. The details of our approach are described in this section.

2.1. CodeT5 backbone model

CodeT5 (Wang et al., 2021) is a state-of-the-art Transformer model pre-trained on a large-scale code-related corpus involving multiple programming languages. It inherits the encoder–decoder architecture from T5 (Raffel et al., 2020), which has been shown beneficial for generation tasks. Moreover, the use of CodeT5 is particularly suitable for our task, as it is able to handle input code snippets from Stack Overflow that may be incomplete and uncompilable. By inputting the code as a sequence of tokens, CodeT5 can understand the structural semantics of the code, thanks to its extensive pre-training on various code understanding objectives. This enables our model to effectively handle the code that are difficult to parse or analyze. We follow the *pre-train then fine-tune* paradigm and further update the trainable parameters θ of CodeT5 on our task-specific dataset D_{so}.

Fine-Tuning: Our objective is to maximize the probability $P_\theta(Y|X)$ given the input code sequence X and the target title Y from the training dataset. X and Y are first split into tokens by the default byte-pair encoding (Gage, 1994) tokenizer of CodeT5, then turned into vectors through the embedding layer. Especially, if the input contains multiple code snippets, we concatenate them to a long sequence with the additional `[NEXT]` identifier. Suppose $X = (x_1 \dots x_{|X|})$ and $Y = (y_1 \dots y_{|Y|})$, where $x_i, y_j \in R^{d_{model}}$. d_{model} is the model hidden size, and $|X|$ and $|Y|$ denote the sequence length with respect to X and Y. We feed X to the encoder, which mainly performs bidirectional self-attention to get

$$C = \text{ENCODER}(X), \tag{1}$$

where $C = (c_1 \dots c_{|X|})$ and vector $c_i \in R^{d_{model}}$ is the hidden representation of the i-th input token. We then feed the auto-regressive decoder with C and Y to get

$$G = \text{DECODER}(C, Y), \tag{2}$$

where $G = (g_1 \dots g_{|Y|})$ and vector $g_j \in R^{d_{model}}$ represents the hidden state of the j-th predicted token. Next, we employ an additional neural layer to map G from the decoder hidden space to the probability distribution over the prediction vocabulary

$$\mathbf{P} = LinearSoftmax(G), \tag{3}$$

where $\mathbf{P} = (P_1 \dots P_{|Y|})$, $P_j \in R^{d_{vocab}}$, d_{vocab} is the vocabulary size, and *LinearSoftmax* is a linear neural network with the *softmax* activation function. Eventually, we can get the loss function for fine-tuning by calculating the average negative log-likelihood

$$Loss = \frac{1}{|Y|} \sum_{j=1}^{|Y|} -\log P_j(y_j), \tag{4}$$

where $P_j(y_j)$ is the predicted probability of the j-th token in the target title.

Inference: We employ the already fine-tuned model and the auto-regressive decoding method to get the predicted title $\hat{Y}$ token by token. To be specific, we first feed the decoder with the start identifier <s> to generate a probability distribution P_1 over the vocabulary, which is used for sampling the first predicted token $\hat{y}_1$. After that, we will again take $(<s>, \hat{y}_1)$ as the input sequence for the decoder to predict the second token $\hat{y}_2$ by repeating the previous steps. Our model predicts each token in $\hat{Y}$ recursively until encountering the ending identifier </s>.

When generating multiple titles, we follow the parallel manner to save the computation cost. Generally, we first take M start identifiers $(<s>_1 \dots <s>_M)^\intercal$ as the input for decoding. In return, we get the sampled first tokens $(\hat{y}_{1,1} \dots \hat{y}_{M,1})^\intercal$ for M candidates. Through the auto-regressive decoding method, our model will repeatedly sample tokens at each step until all the candidates meet the ending identifier </s>. Finally, we will get

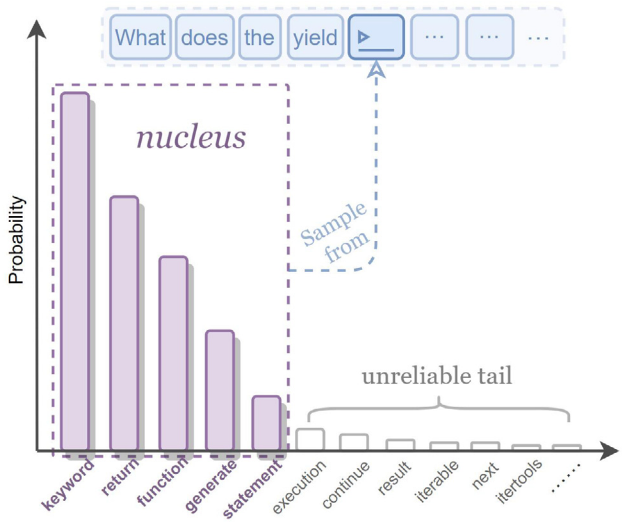


Fig. 3. Applying nucleus sampling to predict the next token in the title.

the sampled titles

$$\hat{Y}_{M,N} = \begin{pmatrix} \hat{y}_{1,1} & \hat{y}_{1,2} & \cdots & \hat{y}_{1,N} \\ \hat{y}_{2,1} & \hat{y}_{2,2} & \cdots & \hat{y}_{2,N} \\ \vdots & \vdots & \ddots & \vdots \\ \hat{y}_{M,1} & \hat{y}_{M,2} & \cdots & \hat{y}_{M,N} \end{pmatrix}, \tag{5}$$

where $\hat{y}_{m,n}$ is the n-th sampled token of the m-th candidate title, N is the length of the longest candidate, and shorter candidates will be padded to length N with a special `[PAD]` identifier.

2.2 Nucleus sampling

An essential step of the decoding step is to sample the predicted token $\hat{y}$ from its probability distribution P over the vocabulary. The most common decoding method is beam search, whose objective is to maximize the probability $P(\hat{Y})$ over the predicted tokens, where $P(\hat{Y}) = \prod_{r=1}^{|\hat{Y}|} P_r(\hat{y}_r)$. Specifically, beam search maintains a fixed number of ordered candidate sequences during decoding, and re-ranks them based on the combinatorial probability of their tokens at each prediction step. The sequence with the highest probability is then selected as the final output. Nonetheless, the content produced by beam search lacks divergence compared with the content written by humans (Holtzman et al., 2019). It is because the maximization-based objective always suppresses the occurrence of uncommon phrases.

In this study, we need to ensure the diversity of generated titles so that they can cover different intentions under varying contexts. To this end, we employ the nucleus sampling (Holtzman et al., 2019) method, the intuition of which is to sample the predicted token from a *nucleus* distribution instead of choosing the token with the highest probability. As shown in Fig. 3, given the already sampled tokens ['What', 'does', 'the', 'yield'], we are now going to choose the next token from the vocabulary distribution. Some tokens in the vocabulary are unlikely to be chosen, such as ['execution', 'continue', …, 'itertools'], which make up the unreliable tail of the distribution. The tokens in the *nucleus*, a minimal subset of the vocabulary that takes up the vast majority of probability mass, are ['keyword', 'return', …, 'statement'], which are most likely to follow the previous token 'yield'. Using nucleus sampling, any token in the *nucleus* has the chance to be chosen, which could bring randomness to the sampling process and significantly improve the diversity of generated titles.

Formally, suppose we are generating the r-th token $\hat{y}_r$ using nucleus sampling, with the probability distribution P_r over the vocabulary V. We first find the minimal *nucleus* set $V^{(p)} \subset V$

$$\sum_{v \in V^{(p)}} P_r(v) \geq \beta, \tag{6}$$

where $v \in V$ and β (also denoted as top-p) is a hyper-parameter of nucleus sampling ranging from 0.0 to 1.0. Let $p' = \sum_{v \in V^{(p)}} P_r(v)$. The original distribution P_r can be re-scaled to

$$P'_r(v) = \begin{cases} P_r(v)/p' & \text{if } v \in V^{(p)} \\ 0 & otherwise, \end{cases} \tag{7}$$

and $\hat{y}_r$ will be sampled from the new distribution P'_r.

2.3 Maximal marginal ranking

Nucleus sampling has been successfully applied to the domain of code generation (Chen et al., 2021a; Fried et al., 2022; Hendrycks et al., 2021; Xu et al., 2022). For example, state-of-the-art code generation models CodeGen (Nijkamp et al., 2022) and OpenAI Codex (Chen et al., 2021a) both incorporate nucleus sampling to generate hundreds and thousands of candidate code solutions for each programming problem, which will significantly improve the problem-solving rates. This can be attributed to the randomness brought by the nucleus sampling, which could enlarge the exploration space of pre-trained models and increase the chance of generating high-quality content. However, due to the random nature of sampling, there is a high variance in generation quality. A common practice to tackle this issue is to sample multiple times and then choose the best samples (Cobbe et al., 2021; Inala et al., 2022; Shi et al., 2022). For example, AlphaCode (Li et al., 2022) employs sophisticated filtering and clustering methods over the generated code solutions to narrow the number of candidates so that the target programming problem can be solved within minimum tries.

In this study, we propose a simple yet effective maximal marginal ranking strategy to ensure the diversity and quality of the final predicted titles. We illustrate the rough idea of our ranking strategy in Fig. 4, where the nodes in the two-dimensional space represent the title samples produced by nucleus sampling. Furthermore, the nodes (titles) that are similar should have a closer distance. Our goal is to find the top-ranked titles with good diversity and quality from all the samples. First, we need to choose a node to start the ranking process. In the example, we choose the node ① from the majority cluster of the red color as the initial candidate. Second, we choose the yellow node ② as another candidate, which has the maximal distance from ① among all the nodes. Then, we choose the blue node ③ as the next candidate, which has the maximal distance from both ① and ②. Similarly, we choose the purple node ④ as the fourth candidate, which has the maximal distance from all the previously chosen nodes. In this way, we can include nodes from different clusters to ensure the diversity of chosen titles. The following introduces the details of our ranking strategy:

Choosing the initial sample: It is crucial to choose a high-quality initial title to start the ranking process because the maximal marginal ranking objective only guarantees the diversity of chosen titles and is blind to their quality. However, discriminating the quality of generated titles is a nontrivial task due to the lack of explicit rules that define 'good quality'. To tackle this problem, we adapt the idea of *self-consistency* (Wang et al., 2022) to facilitate selecting the initial title from generated samples. The *self-consistency* was proposed to improve the performance of reasoning tasks. Generally, after sampling a set of diverse candidates from the model, the final answer should be the one

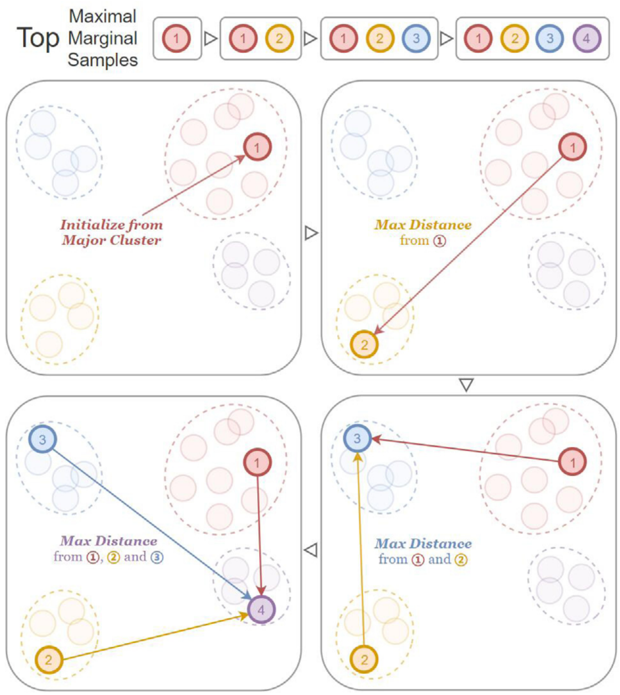


Fig. 4. Illustration of the maximal marginal ranking strategy. Nodes marked in four colors denote the title samples grouped into four clusters based on their distance in the space. This figure shows a four-step example of choosing four titles from all the samples: start from an initial title; the following steps are to choose the title that has the maximal distance to the already chosen ones. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

that is most consistent among the other generated answers. In this study, we propose to measure the quality of generated titles through the *bi*gram consistency. Precisely, we first extract all the token *bi*grams from the generated titles and then calculate the frequency of each *bi*gram. Finally, we rank the titles based on the average frequency of their *bi*grams, and the top-1 title will be considered the most promising initial sample.

Choosing the next samples: Suppose we will offer K titles for the developer, our model needs to sample M candidates for ranking, where $M \gg K$ (e.g., M is 200 when K is 5). In the following notation, $\hat{\mathbf{Y}}_M$ is the set of all M candidate titles and $\hat{\mathbf{Y}}_S$ is a set consisting of the already chosen titles $\hat{Y}_1, \ldots, \hat{Y}_S$, where $\hat{\mathbf{Y}}_S \subset \hat{\mathbf{Y}}_M$ and $S < K$. We now need to choose the next title, $\hat{Y}_{S+1}$, from the remaining candidates $\hat{\mathbf{Y}}_M \setminus \hat{\mathbf{Y}}_S$. To improve the diversity of chosen titles, we propose to find the one that has the maximal distance from those in $\hat{\mathbf{Y}}_S$,

$$\hat{Y}_{S+1} = \underset{\hat{Y}_m \in \hat{\mathbf{Y}}_M \setminus \hat{\mathbf{Y}}_S}{\operatorname{argmax}} \Big(\sum_{\hat{Y}_s \in \hat{\mathbf{Y}}_S} -relevance(\hat{Y}_m, \hat{Y}_s) \Big), \tag{8}$$

where $relevance(\hat{Y}_m, \hat{Y}_s)$ is computed by the cosine similarity of the bag-of-*bi*gram vectors built from the titles. We repeat this process until the size of $\hat{\mathbf{Y}}_S$ reaches K. Additionally, we find that ranking without filtering out stopwords consistently outperforms ranking with stopword removal, regardless of the stopword list used.[2] As a result, we do not include stopword filtering in our ranking process.

[2] We have tried using stopwords from Gensim, NLTK, and scikit-learn.

3 Experimental setup

This section introduces the construction of our dataset, the implementation of our model, the baselines for performance comparison, the automatic evaluation metrics, and the criteria for human evaluation.

3.1 Data preparation

Though previous studies (Gao et al., 2020; Zhang et al., 2022b; Liu et al., 2022a) have proposed open-sourced datasets for the SO title generation task, there are several drawbacks we still have to overcome. Specifically, Gao et al. (2020) only considered the posts with an interrogative title, which account for less than a third of real-world data samples, thus resulting in a biased dataset. While both Zhang et al. (2022b) and Liu et al. (2022a) had their published bi-modal posts stripped and tokenized through natural language processing tools, which damaged the lexical and structural information (such as the white spaces and line breaks) of the code snippets. As a result, we re-construct a large-scale dataset D_{so} to perform our experiments.

D_{so} is built on the *SOTorrent* dataset proposed by Baltes et al. (2018), which is originally used for analyzing the evolution of SO posts. The latest checkpoint of *SOTorrent* contains all the posts from July 2008 to December 2020. Baltes et al. (2018) extracted the code snippets marked by various notations from post bodies and reserved all the white spaces, line breaks, user-defined identifiers, etc. They also removed the noisy fragments wrongly marked as code in the text blocks.

Moreover, previously proposed datasets (Gao et al., 2020; Zhang et al., 2022b; Liu et al., 2022a) only focused on a few dominant Programming Languages (PLs) with abundant data samples, such as *Python*, *C#*, *Java*, *JS(JavaScript)*, and *PHP*. In this study, we consider the posts involving eight PLs, including the above popular ones and the minorities (*C*, *Ruby*, and *Go*). Besides, we perform additional filtering on the collected data to ensure the overall quality of our dataset. We follow the previous settings (Gao et al., 2020; Zhang et al., 2022b; Liu et al., 2022a) and only select the posts satisfying the following four heuristic rules:

1. **The post is not closed;** Stack Overflow may close posts that are not original, relevant to programming and development, or clear and specific enough for users to provide a useful answer.
2. **The post has an accepted answer;** A post that has received an accepted answer is likely to be clear and understandable, as it has been answered satisfactorily according to the original poster or the Stack Overflow community.
3. **The post gets more than one vote;** The number of votes can serve as a measure of perceived value and importance within the Stack Overflow community. A post with more votes may indicate it is useful, relevant, and of high quality.
4. **The post includes code snippets;** For our experiments, it is necessary to ensure that the selected posts include code snippets, as our goal is to generate post titles from the given code snippets.

As for data partitioning, we separate the filtered posts in chronological order, where the latest posts are randomly grouped into validation and test sets, and the rest are for training. This is reasonable because our model should take the past data for training and is applied to new questions in the real-world scenario. We set the number of validation and test samples to 5000 with respect to different PLs. For the languages with insufficient data, we set their proportions of validation and test sets to 10%. In the end, we get the large-scale and high-quality dataset D_{so} for the SO title generation task. The statistics of D_{so} is summarized in Table 1.

Table 1
The number of samples in D_{so} with respect to different PLs.

PL	Train	Validation	Test
Python	190,934	5000	5000
C#	175,070	5000	5000
Java	162,161	5000	5000
JS	151,540	5000	5000
PHP	86,729	5000	5000
C	29,746	3700	3700
Ruby	23,774	3000	3000
Go	6820	850	850
Total	826,774	32,550	32,550

3.2 Implementation details

We implement M_3NSCT5 with the transformers[3] library and the pre-trained model checkpoint[4] of CodeT5, which consists of 12 encoder layers and 12 decoder layers with a hidden size of 768. We optimize all the trainable parameters through AdamW (Loshchilov and Hutter, 2017), with an initial learning rate of 5×10^{-5} scheduled by the linear warm-up. We employ the default byte-pair encoding tokenizer of CodeT5, whose vocabulary size is 32,100. We have two Tesla V100 (16 GB memory) GPUs for training, where each one could hold a data batch size of 8. We further increase the overall batch size to 32 by gradient accumulation. The model is set to train for ten epochs, and we employ the early stopping strategy to avoid overfitting. We follow previous studies in using nucleus sampling with pre-trained models Chen et al., 2021a; Fried et al., 2022; Xu et al., 2022 to set the number of sampled candidates to 200 during decoding. We also tuned the two remaining hyper-parameters of nucleus sampling on our validation dataset and found that a top-p value of 0.8 and a temperature value of 1 provided the best performance. We use these values in the evaluation of our model.

3.3 Baselines

To demonstrate the effectiveness of our approach, we choose several state-of-the-art baseline methods for comparison. We give a brief introduction to these approaches and their experimental settings.

(1) **BM25** was proposed by Robertson and Zaragoza (2009), which has been widely used in the field of information retrieval (Chen et al., 2020b, 2022). It could estimate the relevance of documents for a given search query. The basic idea of BM25 is to rank the referencing documents based on the overlapping query terms, thus ignoring their correlation within the document. Our study adopts this method to retrieve the most relevant posts in the training dataset given the testing code snippets. We could select one or more best matches for each query as the predicted title candidates. We take advantage of the ready-to-use Elasticsearch engine[5] to implement this retrieval baseline, whose default similarity ranking algorithm is BM25.

(2) **Code2Que** was proposed by Gao et al. (2020) to generate SO titles from given code snippets. It is an end-to-end model with the LSTM (Hochreiter and Schmidhuber, 1997) encoder–decoder architecture. Its encoder is a multi-layer bidirectional LSTM network that sequentially handles the input code tokens, while its decoder is a single-layer LSTM that recursively returns the predicted tokens. Moreover, Code2Que incorporates the copy (See et al., 2017) mechanism to allow the decoder to focus on more relevant parts of the input and facilitate capturing some rare but important tokens, and the coverage (Tu et al., 2016) mechanism to discourage generating meaningless repetitions. We employ the OpenNMT[6] library to reproduce this method.

(3) **BART** (Lewis et al., 2020) is a pre-trained Transformer model that achieves state-of-the-art results on a range of NL tasks, especially abstractive summarization, question answering, and machine translation. Unlike the previous successful pre-trained language models BERT (Kenton and Toutanova, 2019) (only with the Transformer encoder) and GPT (Radford et al., 2018) (only with the Transformer decoder), BART employs a standard encoder–decoder architecture and proposes specially designed denoising objectives for pre-training. As a result, BART could improve the performance over previous work when fine-tuned for both text understanding and generation tasks. We reproduce this baseline using its pre-trained model checkpoint.[7]

(4) **CCBERT** was proposed by Zhang et al. (2022b), which is also used for SO title generation but takes bi-modal content (code snippets and text descriptions in the post body) as the model input. CCBERT is a Transformer-based model equipped with CodeBERT (Feng et al., 2020) and an additional copy attention layer. Specifically, CodeBERT is a Transformer encoder pre-trained on a vast scale NL-PL bi-modal corpus, and generate vector representations that support downstream tasks, such as fault prediction and localization (Yu et al., 2019b, 2022; Feng et al., 2021; Yu et al., 2018, 2019a, 2017; Mashhadi and Hemmati, 2021), code clone detection (Zhang et al., 2022a), code smell detection (Liu et al., 2022b), etc. The copy attention layer is an adapted version of the copy mechanism (See et al., 2017) for the Transformer architecture, which helps the model focus on input tokens during decoding. Zhang et al. (2022b) showed the superiority of CCBERT over Code2Que and BART using their collected dataset. We take advantage of their published source code and the pre-trained model checkpoint[8] of CodeBERT to reproduce this baseline.

(5) **SOTitle** was proposed by Liu et al. (2022a), which is another novel approach used for SO title generation. The backbone of SOTitle is the pre-trained T5 (Raffel et al., 2020) model, which follows the Transformer encoder–decoder architecture and employs a transfer learning technique that unifies all text-based language problems into a text-to-text paradigm. T5 was pre-trained on a large-scale corpus crawled from the web and achieved state-of-the-art performance on various NL tasks. Liu et al. (2022a) fine-tuned T5 on their collected SO dataset and reported it could outperform Code2Que and BART. We use their published source code and the pre-trained model checkpoint[9] of T5 to reproduce this baseline.

(6) **PLBART** (Ahmad et al., 2021) is a specialized version of the BART model, whose name is the abbreviation for "Program and Language BART". It also employs the Transformer encoder–decoder architecture and applies denoising objectives for pre-training. PLBART was proposed to produce multilingual representations applicable to NL-PL understanding and generation tasks. It was pre-trained on a large-scale bi-modal corpus collected from GitHub and

3 https://huggingface.co/docs/transformers/index.
4 https://huggingface.co/Salesforce/codet5-base.
5 https://www.elastic.co/elasticsearch/.
6 https://opennmt.net.
7 https://huggingface.co/facebook/bart-base.
8 https://huggingface.co/microsoft/codebert-base.
9 https://huggingface.co/t5-base.

Stack Overflow, then fine-tuned to downstream applications. Results showed that PLBART could outperform state-of-the-art models in a wide range of tasks, especially code summarization and translation. We reproduce this baseline using its pre-trained model checkpoint.[10]

Overall, except BM25, all the baselines are essentially sequence-to-sequence generation models. We notice that CCBERT and SOTitle originally took bi-modal sequences (including code snippets and text descriptions) as input. While in this study, we aim to assist developers who may have difficulty providing clear and concise descriptions for their programming problems on Stack Overflow. Therefore, we use code snippets as the sole input and produce post titles as the output for all the models in our experiments.

3.4 Evaluation methods

We believe a high-quality post title should have good readability and a strong correlation with the post body. Manual evaluation is the ideal way to measure these criteria. Nevertheless, considering the tremendous scale of our dataset, it is necessary to perform an automatic evaluation. An additional human evaluation is performed on a small subset of test samples to demonstrate the intuitive quality of titles generated by our model.

3.4.1 Automatic evaluation

Following previous studies (Gao et al., 2020; Zhang et al., 2022b; Liu et al., 2022a), we automatically evaluate the quality of the generated titles by using two text similarity metrics: BLEU (Papineni et al., 2002) and ROUGE (Lin, 2004). These metrics measure the similarity of a generated title to its corresponding reference title in the original post, with scores ranging from 0 to 1. Higher values indicate greater similarity. In our presentation of the results, we express the scores as percentages for easier readability. For example, a BLEU score of 0.2 is presented as 20%. Below, we provide a more detailed introduction to these metrics.

BLEU originates from machine translation tasks, which mainly calculates the lexical overlap between sentences through n-gram precision. It also incorporates the *brevity penalty* to penalize the behavior of generating short sentences for higher precision scores. In our experiments, we use the BLEU-4 score calculated with 1/2/3/4-gram. Besides, we apply a smoothing method introduced by Lin and Och (2004) to prevent negative scores caused by excessive short sentences. We denote the smoothed method as BLEUS-4 and use the NLTK[11] library for implementation.

ROUGE is a set of metrics commonly used in text summarization, mainly focusing on the n-gram recall. In our experiments, we employ three ROUGE-family metrics, including the ROUGE-1/2 scores that are calculated with 1/2-gram co-occurrence and the ROUGE-L score that concerns the longest common subsequence. Moreover, we use an open source library[12] for implementation.

3.4.2 Human evaluation

In practice, a high-quality post title can be written in different styles. It is hard to tell the actual quality of a generated title based on its similarity with a single reference. Therefore, we perform an additional evaluation on four human-centered criteria. As described in Table 2, each criterion can be quantified by a score number. Specifically, *Readability* measures the grammaticality and fluency of a title, while *Correlation* considers the consistency between a title and its corresponding post body.

Table 2
The criteria used for human evaluation.

Criteria	Scoring standard
Readability	**1** ⇒ Very hard to read and understand **2** ⇒ Just readable and understandable **3** ⇒ Very easy to read and understand
Correlation	**1** ⇒ Totally digress from the key points **2** ⇒ Relevant to the key points **3** ⇒ Exactly match the key points
Diversity	**[1,K]** ⇒ The number of covered intentions
Usability	**1** ⇒ Not useful for post search or writing **2** ⇒ Moderately helpful for post search and writing **3** ⇒ Efficiently streamline post search and writing

Diversity is the number of potential intentions covered by the generated titles. We also evaluate the *Usability* of the generated titles, which reflects their usefulness in helping developers find relevant posts and complete problem descriptions.

For the human evaluators, we recruit six postgraduate students with strong English and programming proficiency. The participants have at least three years of programming experience with their preferred languages, as well as more than one year of experience using Stack Overflow. Additionally, they have at least one year of studying or working experience in English-speaking regions. During the evaluation, participants are evenly divided into three groups according to their preferred programming languages (including the popular *Python* and *Java* languages, as well as the low-resource *Go* language). Finally, we take the average score of each two participants in the same group and report the results by different PLs.

3.4.3 Evaluation on multiple outputs

Finally, we introduce the evaluation method when the model outputs multiple titles for a single input. Suppose the output number is K. We first calculate the scores of all the titles on a specific *Metric* and then take the highest score as the result, which is denoted as *Metric@K*. In this way, we can get the BLEU@K, ROUGE@K, Readability@K, Correlation@K, Diversity@K, and Usability@K that are used for our experiments.

3.5 Research questions

We demonstrate the effectiveness of our model by conducting experiments to answer the following Research Questions (RQs):

RQ-1 **What is the prevalence of the *ambiguity* issue? Motivation:** Fig. 1 illustrates an instance where it is difficult to comprehend the intentions of a specific code snippet without the surrounding context. This research question aims to further examine the prevalence of the *ambiguity* issue in our Stack Overflow title generation task and demonstrate the usefulness of our proposed approach in practical applications.

RQ-2 **Does our approach outperform state-of-the-art baselines under automatic evaluation?**
Motivation: In Section 3.3, we have introduced several state-of-the-art models proposed for the SO title generation task (i.e., Code2Que, CCBERT, and SOTitle) as well as the promising approaches for this task (i.e., BM25, BART, and PLBART). This research question explores whether our model could improve the quality of generated titles compared with the existing methods.

RQ-3 **How effective is our maximal marginal multiple nucleus sampling?**
Motivation: Apart from applying CodeT5 as our backbone, the novelty of M_3NSCT5 mainly lies in our elaborate sampling strategy. We use the nucleus sampling instead of

[10] https://huggingface.co/uclanlp/plbart-base.

[11] http://www.nltk.org/_modules/nltk/translate/bleu_score.html.

[12] https://pypi.org/project/rouge.

Table 3
The automatic evaluation results of M_3NSCT5 and six baselines on the test dataset with respect to different PLs. The values in the table are the average scores expressed as percentages (%), and K is the number of output titles. B4, R1, R2, and RL are the abbreviations of BLEUS-4, ROUGE-1, ROUGE-2, and ROUGE-L.

(a) Python

Setting	Model	B4@K	R1@K	R2@K	RL@K
$K = 1$	BM25	6.95	11.42	1.53	10.70
	Code2Que	12.06	23.07	6.61	22.17
	BART	12.76	24.98	7.56	23.13
	CCBERT	12.98	25.66	8.12	24.15
	SOTitle	12.90	25.45	7.85	23.63
	PLBART	13.05	26.55	8.50	24.69
	M_3NSCT5	**13.34**	**28.65**	**9.68**	**26.44**

(b) C#

Setting	Model	B4@K	R1@K	R2@K	RL@K
$K = 1$	BM25	6.36	9.93	1.85	9.53
	Code2Que	9.91	17.45	5.05	17.55
	BART	11.48	20.95	6.81	19.99
	CCBERT	11.05	20.30	6.79	19.62
	SOTitle	11.52	20.91	6.67	19.98
	PLBART	11.61	22.58	7.73	21.68
	M_3NSCT5	**12.16**	**25.06**	**9.10**	**23.75**

(c) Java

Setting	Model	B4@K	R1@K	R2@K	RL@K
$K = 1$	BM25	6.43	10.68	1.49	10.14
	Code2Que	10.51	19.49	5.24	19.25
	BART	11.53	22.32	6.48	21.11
	CCBERT	11.46	22.13	6.89	21.23
	SOTitle	11.63	22.55	6.58	21.36
	PLBART	11.72	24.14	7.56	22.94
	M_3NSCT5	**12.37**	**26.07**	**8.60**	**24.46**

(d) JavaScript

Setting	Model	B4@K	R1@K	R2@K	RL@K
$K = 1$	BM25	6.60	10.57	1.43	10.03
	Code2Que	11.29	20.83	5.78	20.44
	BART	12.21	23.45	6.68	22.15
	CCBERT	12.36	23.84	7.21	22.63
	SOTitle	12.34	23.73	6.89	22.48
	PLBART	12.50	24.88	7.58	23.65
	M_3NSCT5	**12.74**	**26.96**	**8.53**	**25.25**

(e) PHP

Setting	Model	B4@K	R1@K	R2@K	RL@K
$K = 1$	BM25	7.72	12.15	1.53	11.27
	Code2Que	11.14	19.97	5.14	19.28
	BART	12.24	22.94	5.75	21.29
	CCBERT	12.46	23.04	6.26	21.50
	SOTitle	12.40	22.87	5.76	21.14
	PLBART	12.48	24.06	6.53	22.45
	M_3NSCT5	**12.91**	**25.86**	**7.45**	**23.82**

(f) C

Setting	Model	B4@K	R1@K	R2@K	RL@K
$K = 1$	BM25	6.32	10.07	1.42	9.62
	Code2Que	9.24	16.52	4.22	16.40
	BART	10.68	19.83	5.62	18.97
	CCBERT	10.75	20.15	5.84	19.36
	SOTitle	10.91	20.30	5.69	19.42
	PLBART	10.98	21.67	6.48	20.73
	M_3NSCT5	**11.49**	**24.18**	**7.68**	**22.85**

(g) Ruby

Setting	Model	B4@K	R1@K	R2@K	RL@K
$K = 1$	BM25	6.87	10.98	1.44	10.31
	Code2Que	11.24	20.34	5.72	19.73
	BART	12.74	23.60	7.04	22.08
	CCBERT	12.80	24.36	8.00	23.12
	SOTitle	12.60	23.59	7.13	22.11
	PLBART	12.92	24.41	7.59	22.92
	M_3NSCT5	**13.08**	**26.77**	**9.31**	**25.13**

(h) Go

Setting	Model	B4@K	R1@K	R2@K	RL@K
$K = 1$	BM25	6.74	10.56	1.40	9.87
	Code2Que	10.66	18.84	4.76	19.00
	BART	12.45	22.63	6.44	21.52
	CCBERT	12.54	22.61	7.22	21.76
	SOTitle	12.66	22.70	6.44	21.36
	PLBART	12.82	23.78	7.44	22.84
	M_3NSCT5	**13.21**	**25.57**	**8.85**	**24.50**

beam search and propose the maximal marginal ranking for further performance improvement. This research question aims to investigate the effectiveness of our sampling strategy.

RQ-4 **What is the performance of our approach under human evaluation?**
Motivation: Automatic metrics mainly evaluate the similarity between the generated titles and the given references. Nevertheless, such similarity does not necessarily correlate to human perceptible quality. This research question aims to demonstrate the intuitive quality of generated titles through human evaluation.

4 Results and analysis

4.1 RQ-1: What is the prevalence of the ambiguity issue?

Methods & Results: The meaning of code can be unclear and open to interpretation without sufficient context, which also happens to the code snippets within Stack Overflow posts. To investigate the prevalence of this ambiguity issue, we estimate the percentage of ambiguous code snippets through a human study. Specifically, we ask six human evaluators (introduced in Section 3.4.2) to determine whether they can infer the original intention of a post based on its code snippets. We sample 50 posts from our dataset for each participant according to their preferred programming language and question tags. This results in 300 annotated samples in total covering three PLs (100 in Python, 100 in Java, and 100 in Go). The results show that **43%**, **37%**, and **51%** of the posts are marked as ambiguous for Python, Java, and Go, respectively. The evaluators conclude several factors that contribute to the ambiguity of a code snippet, including its length (too short or too long), the absence of comments, and the differing perspectives and prior experiences of the evaluators. These findings indicate a significant prevalence of the *ambiguity* issue, which contributes to the usability of our proposed approach in real-world applications.

> Answer to RQ-1: The *ambiguity* issue is prevalent in our Stack Overflow title generation task, which highlights the need and usefulness of our proposed approach.

4.2 RQ-2: Does our approach outperform state-of-the-art baselines under automatic evaluation?

Methods: We compare M_3NSCT5 with six state-of-the-art baselines on the four automatic evaluation metrics (i.e., BLEUS-4, ROUGE-1, ROUGE-2, and ROUGE-L). Since we propose to overcome the *ambiguity* issue by sampling multiple times, we also experiment with the number of outputs $K = 3$ and $K = 5$. Following previous studies (Zhang et al., 2022b; Liu et al., 2022a) on Stack Overflow title generation, we train all the models, except for BM25, on the whole training set of our D_{so} dataset that covers

Table 4
The automatic evaluation results of M_3NSCT5, PLBART, and BM25 on the test dataset when $K > 1$, where K is the number of output titles. The values in the table are the average scores expressed as percentages (%) of the best title among the K title candidates. B4, R1, R2, and RL are the abbreviations of BLEUS-4, ROUGE-1, ROUGE-2, and ROUGE-L.

(a) Python

Setting	Model	B4@K	R1@K	R2@K	RL@K
$K = 3$	BM25	11.18	19.26	3.56	17.95
	PLBART	15.11	30.81	10.76	28.67
	M_3NSCT5	**15.94**	**33.42**	**11.93**	**30.96**
$K = 5$	BM25	12.72	22.55	4.87	21.00
	PLBART	16.17	33.09	12.06	30.86
	M_3NSCT5	**17.08**	**35.58**	**13.28**	**33.05**

(b) C#

Setting	Model	B4@K	R1@K	R2@K	RL@K
$K = 3$	BM25	10.81	17.37	4.07	16.66
	PLBART	14.44	27.02	9.89	25.84
	M_3NSCT5	**14.87**	**29.54**	**10.87**	**27.97**
$K = 5$	BM25	12.52	20.85	5.50	19.92
	PLBART	15.57	29.29	11.26	28.00
	M_3NSCT5	**16.06**	**31.75**	**12.14**	**30.09**

(c) Java

Setting	Model	B4@K	R1@K	R2@K	RL@K
$K = 3$	BM25	10.67	18.12	3.24	17.14
	PLBART	14.47	28.25	9.42	26.73
	M_3NSCT5	**14.99**	**30.71**	**10.49**	**28.89**
$K = 5$	BM25	12.41	21.60	4.56	20.46
	PLBART	15.47	30.45	10.61	28.86
	M_3NSCT5	**16.21**	**32.94**	**11.80**	**30.99**

(d) JavaScript

Setting	Model	B4@K	R1@K	R2@K	RL@K
$K = 3$	BM25	10.87	18.02	3.33	17.02
	PLBART	14.69	29.44	9.68	27.74
	M_3NSCT5	**15.15**	**31.46**	**10.46**	**29.53**
$K = 5$	BM25	12.49	21.19	4.50	20.01
	PLBART	15.80	31.68	11.01	29.84
	M_3NSCT5	**16.25**	**33.64**	**11.72**	**31.65**

(e) PHP

Setting	Model	B4@K	R1@K	R2@K	RL@K
$K = 3$	BM25	12.08	19.93	3.46	18.38
	PLBART	14.67	28.99	8.61	26.96
	M_3NSCT5	**15.66**	**31.75**	**9.81**	**29.32**
$K = 5$	BM25	13.75	23.39	4.93	21.60
	PLBART	15.76	31.16	9.75	29.04
	M_3NSCT5	**16.97**	**34.53**	**11.46**	**31.84**

(f) C

Setting	Model	B4@K	R1@K	R2@K	RL@K
$K = 3$	BM25	10.72	17.61	3.28	16.76
	PLBART	13.62	26.33	8.49	25.09
	M_3NSCT5	**14.09**	**28.79**	**9.48**	**27.21**
$K = 5$	BM25	12.49	20.96	4.49	19.83
	PLBART	14.78	28.61	9.73	27.25
	M_3NSCT5	**15.40**	**31.22**	**10.73**	**29.46**

(g) Ruby

Setting	Model	B4@K	R1@K	R2@K	RL@K
$K = 3$	BM25	11.04	18.28	3.40	17.18
	PLBART	15.24	29.63	10.34	27.86
	M_3NSCT5	**16.17**	**32.16**	**11.49**	**30.25**
$K = 5$	BM25	12.84	21.82	4.81	20.51
	PLBART	16.98	31.90	11.84	30.06
	M_3NSCT5	**17.63**	**34.77**	**13.14**	**32.70**

(h) Go

Setting	Model	B4@K	R1@K	R2@K	RL@K
$K = 3$	BM25	11.56	18.30	3.42	17.30
	PLBART	15.26	28.58	9.19	27.13
	M_3NSCT5	**16.22**	**30.80**	**10.97**	**29.39**
$K = 5$	BM25	13.11	21.58	4.71	20.33
	PLBART	16.48	30.64	10.23	29.12
	M_3NSCT5	**17.69**	**33.60**	**12.23**	**31.74**

eight PLs and test on individual subsets of different PLs. The experimental results are shown in Tables 3 and 4.

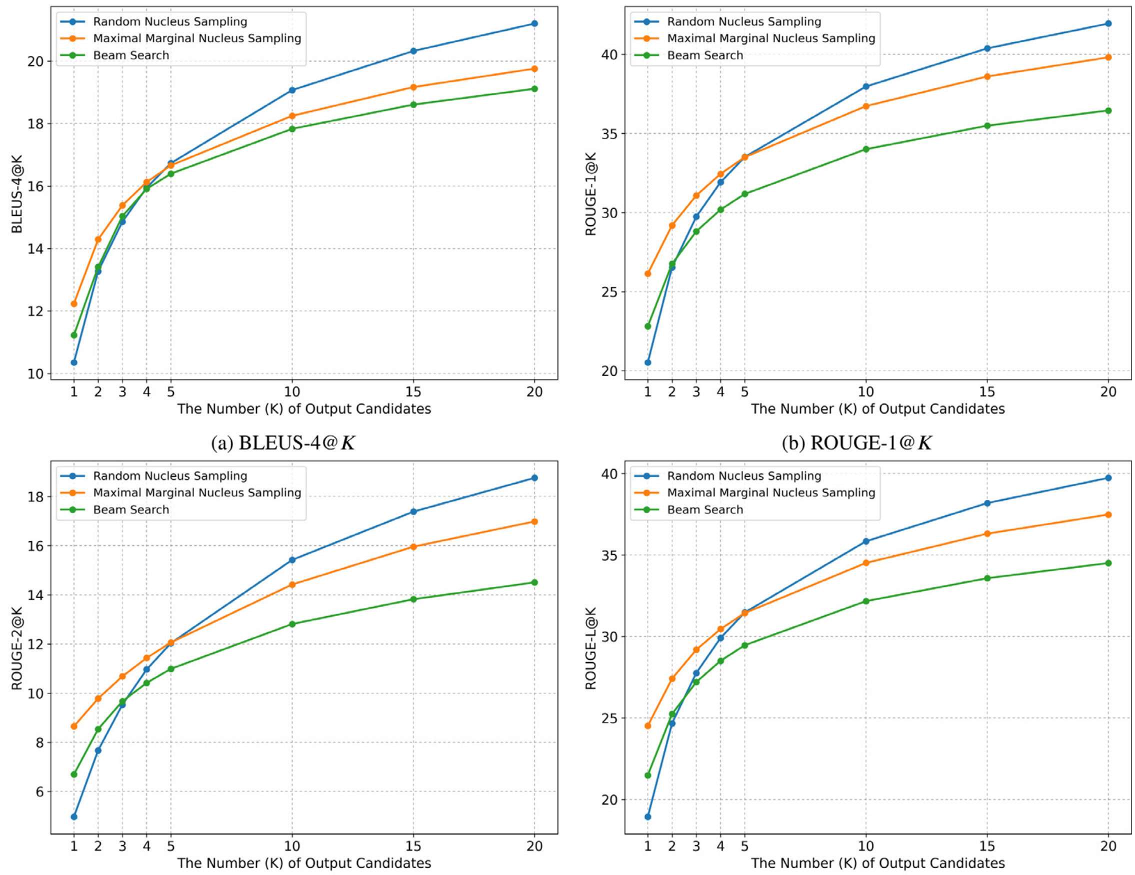
Results: Based on the results, we can conclude that M_3NSCT5 achieves the best performance under automatic evaluation, outperforming all the baseline models. Specifically, we have the following findings:

(1) According to Table 3, when all the models output only one candidate (i.e., $K = 1$), M_3NSCT5 could achieve the best performance. Among all the baselines, the retrieval method BM25 obtains the worst performance. The LSTM-based Code2Que outperforms BM25 by a large margin but is no match for large pre-trained Transformer models, which aligns with the results of the previous study (Liu et al., 2022a). We also find that BART, CCBERT, and SOTitle share similar results on different PL subsets, where all of them are worse than PLBART. We attribute the good performance of PLBART to its generation-oriented denoising objectives and code-related corpus for pre-training. Furthermore, M_3NSCT5 outperforms PLBART by 3.3%, 8.8%, 16.5%, and 7.9% in terms of BLEUS-4, ROUGE-1, ROUGE-2, and ROUGE-L on average of different PL subsets.

(2) According to Table 4, when all the models output multiple candidates ($K = 3$ and $K = 5$), M_3NSCT5 could also significantly improve the performance as well as outperform other baselines. We choose BM25 for comparison because returning multiple candidates for a query is common for information retrieval. We also compare with PLBART, which shares a similar model architecture with CodeT5 and has the most competitive results when $K = 1$.

As shown in Table 4, increasing the output number K (i.e., offering more title candidates for the developers to choose from) boosts the performance of all models by a large margin. In particular, when K changes from 1 to 3, M_3NSCT5 performs 21.5%, 18.9%, 23.7%, and 19.1% better in terms of BLEUS-4, ROUGE-1, ROUGE-2, and ROUGE-L on average of different PL subsets. As for the baselines, BM25 remains the worst performance. Though PLBART achieves acceptable results, M_3NSCT5 still outperforms it by 4.7%, 8.6%, 12%, and 8.1% in terms of BLEUS-4, ROUGE-1, ROUGE-2, and ROUGE-L on average when $K = 3$, and by 4.9%, 8.6%, 11.7%, and 7.9% in terms of BLEUS-4, ROUGE-1, ROUGE-2, and ROUGE-L on average when $K = 5$.

(3) According to the sub-tables of different PLs in Tables 3 and 4, M_3NSCT5 achieves excellent performance on every PL subset. Surprisingly, the results on less popular PLs like *C*, *Ruby*, and *Go* are totally comparable to the dominant ones. This finding also applies to other baselines, where models never pre-trained on code-related corpus like Code2Que, BART and SOTitle can perform well on all subsets. Though PLBART has only been pre-trained on *Python* and *Java* corpus, it can achieve quite competitive results to our model on other PLs. All the evidence indicates that models can benefit from fine-tuning on the joint dataset of different PLs, and it is applicable to introduce new PLs with less sufficient data to the SO title generation task.



(a) BLEUS-4@K (b) ROUGE-1@K

(c) ROUGE-2@K (d) ROUGE-L@K

Fig. 5. Automatic evaluation results of equipping the fine-tuned CodeT5 model with three sampling strategies.

Table 5
Human evaluation results of M_3NSCT5, PLBART, and BM25 on three programming languages when $K = 3$. S-1, S-2, and S-3 represent the percentage of samples rated to score 1, 2, and 3. S-Avg represents the average score of title candidates. The numbers in the table are the mean values of two participants in the same group.

Language	Criteria(@3)	M_3NSCT5				PLBART				BM25			
		S-1	S-2	S-3	S-Avg	S-1	S-2	S-3	S-Avg	S-1	S-2	S-3	S-Avg
Python	Readability	–	23%	77%	2.77	–	24%	76%	2.76	–	13%	87%	**2.87**
	Diversity	3%	33%	64%	**2.61**	17%	49%	34%	2.17	5%	35%	60%	2.55
	Correlation	11%	52%	37%	**2.26**	19%	49%	32%	2.13	68%	32%	–	1.32
	Usability	1%	54%	45%	**2.44**	12%	72%	16%	2.04	71%	29%	–	1.29
Go	Readability	–	39%	61%	2.61	–	38%	62%	2.62	–	16%	84%	**2.84**
	Diversity	4%	39%	57%	2.53	21%	51%	28%	2.07	3%	37%	60%	**2.57**
	Correlation	12%	55%	33%	**2.21**	19%	54%	27%	2.08	71%	29%	–	1.29
	Usability	4%	59%	37%	**2.33**	14%	70%	16%	2.02	72%	28%	–	1.28
Java	Readability	–	31%	69%	2.69	–	32%	68%	2.68	–	14%	86%	**2.86**
	Diversity	3%	36%	61%	**2.58**	18%	49%	33%	2.15	4%	36%	60%	2.56
	Correlation	10%	56%	34%	**2.24**	19%	51%	30%	2.11	69%	31%	–	1.31
	Usability	3%	59%	38%	**2.35**	13%	69%	18%	2.05	72%	28%	–	1.28

> Answer to RQ-2: Our proposed M_3NSCT5 can outperform state-of-the-art baselines on automatic evaluation by generating titles of higher quality under different experimental settings.

4.3 RQ-3: How effective is our maximal marginal multiple nucleus sampling?

Methods: Remaining the already fine-tuned CodeT5 unchanged, we compare the performance of the three decoding strategies when K varies from 1 to 20. First, we use the vanilla Beam Search (BS) method, which ranks the generated candidates by the combinatorial probability of their tokens. We set the beam size to 20 and select the top K candidates as output. Second, we repeat Random Nucleus Sampling (RNS) K times, then take the sampled candidates as output. The third is our proposed strategy, which mainly performs Maximal Marginal Nucleus Sampling (MMNS). The experimental results are shown in Fig. 5, where the sub-figures individually demonstrate the performance of the three sampling strategies on each automatic evaluation metric, averaged on eight PL subsets.

Results: From the results, we can easily find the superiority of our sampling strategy, especially when $K \leq 5$. Specifically, we have the following findings:

(1) The performance of RNS has a large fluctuation range and is highly sensitive to the value of K. While BS has a more moderate performance improvement than RNS when K increases. Considering the user scenario of the SO title generation task, it is applicable when K takes small values, e.g., recommending at most 5 title candidates for the developer. When $K \leq 2$, BS outperforms RNS, indicating the titles generated with higher probability are of better quality. When $3 \leq K \leq 5$, BS performs worse than RNS, showing that ranking titles purely on probability could damage the trait of diversity.
(2) Our MMNS strategy takes advantage of nucleus sampling and maximal marginal ranking to increase the diversity of output titles. It also incorporates self-consistency voting to obtain the most promising title with higher quality. As a result, when $K = 1$, MMNS outperforms BS (and RNS) by 9%, 14.6%, 29.2%, and 14.1% (by 18.2%, 27.4%, 74%, and 29.5%) in terms of BLEUS-4, ROUGE-1, ROUGE-2, and ROUGE-L. When $K = 3$, MMNS outperforms RNS (and BS) by 3.5%, 4.5%, 12.1%, and 5.1% (by 2.3%, 7.9%, 10.5%, and 7.3%) in terms of BLEUS-4, ROUGE-1, ROUGE-2, and ROUGE-L. When $K = 5$, MMNS is still comparable to RNS.

Answer to RQ-3: Our sampling strategy can improve the quality and diversity of generated titles, especially when $K \leq 5$, making it more suitable and effective for the SO title generation task.

4.4 RQ-4: What is the performance of our approach under human evaluation?

Methods: As introduced in Section 3.4.2, we recruit six students for human evaluation. Participants are required to score the titles generated by M_3NSCT5, PLBART, and BM25 on three PLs. We assign each participant 100 random post samples where each post is paired with nine titles generated by the three approaches (i.e., the output number $K = 3$). Participants do not know the titles are generated by which approach during the evaluation, and they are requested to tell the scores of each sample according to the scoring standard in Table 2. Additionally, we employ Cohen's Kappa (Cohen, 1960) to measure the agreement between the two participants in each group. The main evaluation results are shown in Table 5 and the Cohen's Kappa statistics are summarized in Tables 6 and 7. In addition, we provide examples in Table 8 to illustrate the quality and diversity of the titles generated by the three approaches. We also include the automatic and human evaluation scores of these generated titles in Table 9 to provide a more intuitive understanding of our evaluation.

Results: From the results, we can find that M_3NSCT5 achieves the best performance in terms of *Diversity*, *Correlation*, and *Usability* compared with the other two approaches. Furthermore, the performance of M_3NSCT5 is consistent among different programming languages, even for the low-resource *Go* subset. Moreover, the participants have a substantial or nearly perfect agreement according to Cohen's Kappa statistics, which validates the trustworthiness of our human evaluation. Specifically, we have the following findings:

(1) In terms of the *Readability* criterion, BM25 achieves the best performance because it just returns the human-written titles retrieved from the training set. Besides, both M_3NSCT5 and PLBART can achieve a competitive score ≥2.6 on three PLs, indicating the capability of pre-trained models on natural language generation. The examples in Table 8 also demonstrate the good readability of generated titles.

Table 6
The interpretation of Cohen's Kappa agreement.

Cohen's Kappa	Interpretation
0%	No agreement
1%∼20%	Slight agreement
21%∼40%	Fair agreement
41%∼60%	Moderate agreement
61%∼80%	Substantial agreement
81%∼99%	Near perfect agreement
100%	Perfect agreement

(2) In terms of the *Diversity* criterion, both M_3NSCT5 and BM25 have good results, having an average number of distinct titles ≥2.5 when $K = 3$. BM25 performs well because there are almost no duplicate posts in the training set. The excellent performance of M_3NSCT5 should attribute to our elaborate sampling strategy that maximizes the difference between the output titles. In contrast, the poor performance of PLBART on *Diversity* should blame on the beam search decoding method. From the examples in Table 8, we can find that the titles generated by PLBART have higher lexical and semantic overlap than our approach.
(3) As for the *Correlation* criterion, M_3NSCT5 outperforms PLBART and BM25, with around 90% samples relevant to or exactly matching the key points of original posts. It shows the feasibility of inferring user intents from code snippets. We attribute the excellent performance of M_3NSCT5 to the initial choice of high-quality titles based on self-consistency when performing the maximal marginal ranking. We can see from Tables 8 and 9 that BM25 is totally off the point and even recommends *PHP* posts for the *Go* code snippets because of their high lexical overlap, indicating the limitations of the retrieval method. The titles generated by PLBART are relevant to the posts but still missing the points. At the same time, M_3NSCT5 shows a good ability to understand the code snippets and generate diverse title candidates, with the best candidate closely related to the post.
(4) Regarding the *Usability* criterion, results in Table 5 demonstrate that our M_3NSCT5 significantly outperforms the PLBART and BM25 baselines in terms of generating useful titles for developers. Around 40% of the titles generated by M_3NSCT5 are found to significantly assist human evaluators in retrieving relevant posts and completing their problem descriptions, while less than 5% are marked as useless. This suggests that M_3NSCT5 has strong potential for real-world applications. In contrast, BM25 obtains the worst performance, with around 70% of its titles being deemed useless by evaluators. This is likely due to the low correlation scores and relevance to the input code of these titles. While PLBART performs better than BM25 in terms of usability, it is still not as effective as M_3NSCT5. Therefore, the findings validate the effectiveness of M_3NSCT5 in our Stack Overflow title generation task.

Answer to RQ-4: Our approach shows a strong ability to generate high-quality and diverse post titles that can help developers retrieve relevant posts and complete their problem descriptions.

5 Limitation

While our proposed approach, M_3NSCT5, has demonstrated impressive performance in generating high-quality and diverse

Table 7
The agreement values of human evaluation results.

Language	Criteria(@3)	M_3NSCT5	PLBART	BM25
Python	Readability	83.1%	78.3%	73.7%
	Diversity	79.4%	83.9%	84.5%
	Correlation	79.6%	84.0%	81.8%
	Usability	84.3%	82.0%	85.5%
Go	Readability	79.2%	74.9%	85.2%
	Diversity	81.0%	77.6%	80.3%
	Correlation	82.7%	76.9%	76.0%
	Usability	80.7%	74.5%	80.3%
Java	Readability	76.9%	81.8%	83.4%
	Diversity	80.1%	77.7%	84.4%
	Correlation	85.8%	80.6%	86.0%
	Usability	80.4%	83.2%	80.3%

Stack Overflow post titles from input code snippets, there are still some limitations to consider. One limitation is that, like all data-driven deep learning models, M_3NSCT5 is susceptible to the long-tail problem and may not perform well on the code snippets outside the scope of training data. Additionally, M_3NSCT5 may struggle to handle input code that is either too short or too long, as this can lead to a lack of semantics or an increase in semantic complexity, respectively. As a result, it may be difficult for M_3NSCT5 to generate titles with the expected intentions in these cases, even if the diversity of generated titles is high. Despite these limitations, we believe that the potential benefits of using M_3NSCT5 in real-world applications outweigh the potential risks, and our work will inspire future research in this area.

6 Related work

A Stack Overflow post usually consists of three parts: body, title, and tags. Previous studies on the tag recommendation task demonstrated that one could utilize the recommended tags for post retrieval, similar to our motivation. But the discrete tags are more suitable for post classification than serving as search queries. In comparison, a coherent and informative post title can better help developers understand the problem and search for related posts. Once developers have narrowed their search to specific posts, answer summarization techniques can be helpful in identifying and extracting the most relevant information from lengthy responses. Moreover, generating post titles from code snippets can be seen as a specialized PL-to-NL translation task. A closely related task is code summarization, an emerging research direction in software engineering and natural language processing. This section introduces the previous studies of post title generation, tag recommendation, answer summarization, and the recent advances in code summarization.

Title generation: Gao et al. (2020) first proposed the SO title generation task to help improve the quality of poorly written question posts. They trained an LSTM network with the copy (See et al., 2017) and coverage (Tu et al., 2016) mechanisms to generate titles from mined code snippets. Later, Zhang et al. (2022b) and Liu et al. (2022a) found that taking advantage of both code snippets and text descriptions in the post body could significantly improve the quality of generated titles. Though utilizing the natural language descriptions could reduce the ambiguity of the context, it is less helpful when developers cannot provide good question descriptions. Therefore, in this study, we focus on the application scenario where only code snippets are available. We propose M_3NSCT5 to improve the quality and diversity of generated titles compared with previous approaches.

Tag recommendation: Post tags are vital for Stack Overflow, which are helpful in organizing relevant posts. Nevertheless, poorly chosen tags may cause severe redundancy over time. To tackle this challenge, early studies (Xia et al., 2013; Wang et al., 2014, 2015; Zhou et al., 2017; Liu et al., 2018) proposed to automatically recommend tags with the given post body, title, and user profile through feature extraction and similarity-based methods. Recently, Zhou et al. (2019) and Xu et al. (2021) introduced deep learning models for this task, which could achieve better performance. Moreover, Devine and Blincoe (2022) and He et al. (2022b) proposed to take advantage of pre-trained models for further improvement. However, it remains unexplored to recommend post tags with code snippets. We believe tag recommendation and title generation models can be combined for post retrieval, which is a direction of our future work.

Answer summarization: Stack Overflow is a popular resource for developers seeking answers to their questions, but the process of reviewing related posts and detailed answers can be time-consuming. To address this issue, researchers proposed many automated answer summarization approaches to improve the efficiency and effectiveness of the information-seeking process. For example, Xu et al. (2017) developed the AnswerBot framework to generate answer summaries for non-factoid technical questions. Nadi and Treude (2020) studied the problem of identifying essential sentences in Stack Overflow answers. Chatterjee (2022) utilized natural language based approaches to suggest code pairs from a post. More recently, Kou et al. (2022) and Chengran et al. (2022) proposed more large scale datasets and promising methods for the answer summarization task. Empirical experiments have demonstrated the effectiveness of these approaches. And we believe the answer summarization approaches can be combined with our title generation model to further improve the user experience on Stack Overflow.

Code summarization: The code summarization task is to generate readable descriptions of the given program, aiming to save the effort of developers on program comprehension. Nowadays, with the emergence of large-scale NL-PL bi-modal datasets and the rapid development of deep learning techniques (Chen and Lu, 2020; He et al., 2022a; Chen et al., 2020a; He et al., 2021; Yang et al., 2021a; He et al., 2020; Zhao et al., 2022, 2023), it becomes feasible to train deep models to generate high-quality code summaries. Most existing code summarization approaches (Wan et al., 2018; Ahmad et al., 2020) follow the encoder-decoder framework, which first propose to encode the sequential code tokens into a hidden state via LSTM or Transformer, and then decode it into a natural-language summary. Then, a number of follow-up studies (Tang et al., 2021, 2022; Shi et al., 2021; Guo et al., 2022b; Zhou et al., 2022a; Li et al., 2021; Cheng et al., 2022; Zhou et al., 2022b; Yang et al., 2021b) proposed to incorporate the structural information by parsing the source code into abstract syntax trees or control flow graphs to improve the performance. Since code snippets extracted from SO posts are always problematic, we cannot apply static parsing techniques to get the syntax trees or control flow graphs. Therefore, as mentioned in Section 3.1, we try to reserve the structural information by keeping the white spaces and line breaks in code snippets. Furthermore, some other studies (Lu et al., 2021; Feng et al., 2020; Ahmad et al., 2021; Wang et al., 2021; Guo et al., 2022a; Fried et al., 2022) proposed to utilize the large-scale unlabeled data and self-supervised learning to pre-train models through self-supervised objectives and achieved state-of-the-art results on code summarization benchmarks. Motivated by the good performance of pre-training, we employ the pre-trained CodeT5 model as our backbone. Nonetheless, generating titles for SO posts presents its

Table 8
Example input code snippets and the titles generated by M$_3$NSCT5, PLBART, and BM25. The scores of the generated titles on automatic evaluation metrics and human criterion are shown in Table 9.

Code Snippets	Titles

Example 1 — Python Language:
`letter_list = ['a','d','o','m','s']`
`>>> df`
`ID WORD`
`1 'yellow'`
`2 'orange'`
`3 'green'`
`4 'blue'`
`>>> expected output`
`ID WORD`
`3 'green'`
`4 'blue'`
	Origin:
	<u>filter dataframe for words which do not contain any of the letters in a list</u>
	M$_3$NSCT5:
	1.1 how to filter rows in pandas dataframe based on a list of letters
	1.2 pandas - replace rows with values of list from another list
	1.3 pandas : groupby by letters of string
	PLBART:
	1.4 use pandas to group words from a list
	1.5 how to group elements of a list in pandas
	1.6 how to group by a second list in pandas
	BM25:
	1.7 pandas merge creates unwanted duplicate entries
	1.8 using nested loops in order to group data in a csv
	1.9 sort a list with known values before unknown values

Example 2 — Java Language:
`try(Scanner scan = new Scanner(System.in)) {`
`    String s = scan.nextLine();`
`    System.out.println(s);`
`    scan = null;`
`}`
	Origin:
	<u>try with resource why cannot modify resource</u>
	M$_3$NSCT5:
	2.1 how to close a scanner in a try-with-resources block
	2.2 why is the scanner null after the first line
	2.3 java scanner.nextline() not working as expected
	PLBART:
	2.4 will it be a nullpointerexception in try-with-resource
	2.5 java scanner() throws nullpointerexception
	2.6 how do I prevent a nullpointerexception from being thrown
	BM25:
	2.7 copying characters in a string
	2.8 java how to parse for words in a string for a specific word
	2.9 how to terminate scanner when input is complete

Example 3 — Go Language:
`{{range $m := .messages}}`
`    <div>Message subject: {{$m.Subject}}</div>`
`    {{$lastMsg := $m}}`
`{{end}}`
`<div>`
`    The last message's subject: {{$lasMsg.Subject}}`
`</div>`
 	
`undefined variable "$lastMsg"`
 	
`unexpected ":=" in operand`
	Origin:
	<u>how to use template variable outside a loop</u>
	M$_3$NSCT5:
	3.1 how to access a variable in a range in go template
	3.2 golang nested loop variable not found
	3.3 why does {{$variable}} not work in this case
	PLBART:
	3.4 go range variables in html template
	3.5 go templates, use variable in range
	3.6 accessing a slice value inside a range in go
	BM25:
	3.7 rails: form in partial for new nested resource
	3.8 how to use visual studio - generated async wcf calls
	3.9 how do i secure this php script

own challenges, as the code in a post may be flawed or incomplete, making it difficult to understand the intended semantics. Additionally, developers may have different intentions for the same code, which highlights the need to increase the diversity of generated titles.

7 Threats to validity

This section reveals the potential threats that may affect the reproduction of our experiments and the validation of our results.

The threats to internal validity mainly relate to the implementation of baseline models. For CCBERT and SOTitle that already have released source code and model checkpoints, we keep their default hyper-parameters unchanged in our experiments. For BM25, Code2Que, BART, and PLBART, which have no off-the-shelf implementations, we take advantage of the widely used libraries (i.e., Elasticsearch, OpenNMT, and transformers) for reproduction and tune their hyper-parameters to the best on our dataset. In this way, we make sure the comparison between our model and the baselines is fair. And we release our implementations of the baselines to facilitate future studies.

The threats to external validity mainly relate to the construction of our dataset. We have tried our best to ensure the quality of our dataset. First, we utilize the already processed dataset SOTorrent to avoid potential bugs when extracting code snippets from the post body. Second, we only include the filtered high-quality posts in the dataset to reduce the noise of training and test data. Third, our dataset covers eight programming languages, including the minorities (*Ruby* and *GO*), which would better test the generalization ability of models. Moreover, we split the train and test sets in chronological order to fit real-world scenarios. We also release our dataset for validation and reproduction.

The threats to construct validity mainly relate to the evaluation methods. Though BLEU and ROUGE are the most popular evaluation metrics for generation tasks, measuring the quality of the generated content remains an open challenge. In the SO title generation task, there is no golden title for a given post, which makes it unfair to judge the quality of generated titles by comparing them with the only reference title. Therefore, we perform an additional human evaluation to show the intuitive quality of generated titles. To perform a comprehensive study, we invite six participants to evaluate the posts covering three programming languages. Additionally, we conducted a large-scale automated evaluation of readability using the classical Automated Readability Index (ARI) (Senter and Smith, 1967) and Flesch Reading Ease (FRE) (Kincaid et al., 1975) indices on our test dataset, which consists of high-quality posts that could attract significant attention and elicit responses. However, the results of the readability analysis using ARI and FRE showed that the human-written titles in our test dataset were perceived as difficult to read. The SO titles

Table 9
The detailed scores of the generated titles listed in Table 8. The scores of automatic evaluation metrics are presented as percentages (%), and the scores of human evaluation are the averaged numbers of two human evaluators in a group. The three titles generated by the same model share the *Diversity* score for each example.

Model	Title No.	Automatic Evaluation				Human Evaluation			
		BLEUS-4	ROUGE-1	ROUGE-2	ROUGE-L	Readability	Diversity	Correlation	Usability
Example 1 — Python Language									
M_3NSCT5	*1.1*	13.60	50.00	7.69	28.57	3.0	3.0	2.5	3.0
	1.2	8.36	15.38	0.00	16.00	3.0	3.0	2.0	2.0
	1.3	8.36	18.18	0.00	9.09	2.5	3.0	1.5	1.5
PLBART	*1.4*	11.00	26.09	9.52	26.09	3.0	2.0	1.5	1.5
	1.5	11.82	33.33	9.09	25.00	3.0	2.0	1.5	1.5
	1.6	9.25	25.00	0.00	16.67	3.0	2.0	2.0	1.5
BM25	*1.7*	0.00	0.00	0.00	0.00	3.0	3.0	1.0	1.5
	1.8	9.94	15.38	8.33	16.00	3.0	3.0	1.0	1.0
	1.9	9.94	16.67	9.09	17.39	2.5	3.0	1.0	1.0
Example 2 — Java Language									
M_3NSCT5	*2.1*	8.15	20.00	0.00	23.53	3.0	3.0	2.5	2.5
	2.2	12.14	12.50	0.00	14.29	3.0	3.0	2.0	2.0
	2.3	0.00	0.00	0.00	0.00	2.5	3.0	1.5	1.5
PLBART	*2.4*	12.00	33.33	0.00	37.50	3.0	2.0	2.0	1.5
	2.5	0.00	0.00	0.00	0.00	3.0	2.0	2.0	1.5
	2.6	0.00	0.00	0.00	0.00	3.0	2.0	2.0	1.5
BM25	*2.7*	0.00	0.00	0.00	0.00	3.0	3.0	1.0	1.0
	2.8	0.00	0.00	0.00	0.00	2.5	3.0	1.0	1.0
	2.9	0.00	0.00	0.00	0.00	3.0	3.0	1.5	1.5
Example 3 — Go Language									
M_3NSCT5	*3.1*	16.97	52.63	11.76	47.06	3.0	3.0	3.0	3.0
	3.2	16.52	28.57	0.00	14.29	3.0	3.0	2.5	2.5
	3.3	7.43	9.52	0.00	10.53	3.0	3.0	2.0	2.0
PLBART	*3.4*	13.89	14.29	0.00	14.29	2.5	2.0	2.0	2.5
	3.5	16.52	26.67	0.00	26.67	2.5	2.0	2.0	2.0
	3.6	12.26	11.76	0.00	12.50	2.5	2.0	2.5	2.5
BM25	*3.7*	0.00	0.00	0.00	0.00	2.5	3.0	1.0	1.0
	3.8	22.28	33.33	25.00	33.33	3.0	3.0	1.0	1.0
	3.9	13.89	13.33	0.00	13.33	2.5	3.0	1.0	1.0

often contain code identifiers and special symbols, and are often very short, making them difficult to evaluate using indices such as ARI and FRE, which often assume complete, grammatically correct sentences. Based on the finding, these indices may not be suitable for evaluating the readability of the generated titles in our task.

8 Conclusion and future work

In this paper, we proposed M_3NSCT5, a novel approach to automatically generate Stack Overflow post titles from the given code snippets, which can help non-English speaking or inexperienced developers improve their poorly written question posts. Combining the pre-trained CodeT5 model and the maximal marginal multiple nucleus sampling strategy, M_3NSCT5 can generate high-quality and diverse title candidates for the developers to choose from. To validate the effectiveness of our approach, we have built a large-scale dataset with 890,000 posts covering eight programming languages and choose six state-of-the-art baselines for comparison. We performed extensive experiments to demonstrate the superiority of our approach, including an automatic evaluation on the BLEU and ROUGE metrics and a human evaluation using the *Readability*, *Correlation*, *Diversity*, and *Usability* criteria. Results showed that M_3NSCT5 outperforms all the baseline methods by a significant margin and has great potential for real-world applications.

For future work, we plan to further incorporate more powerful pre-trained language models and tag recommendation methods to improve the title generation task performance. Moreover, we plan to deploy our model as web services so that real-world developers from Stack Overflow could benefit from our work and produce valuable user feedback for future improvement.

CRediT authorship contribution statement

Fengji Zhang: Investigation, Data curation, Writing – original draft. **Jin Liu:** Funding acquisition, Supervision. **Yao Wan:** Formal analysis, Writing – review & editing. **Xiao Yu:** Methodology, Validation, Supervision. **Xiao Liu:** Methodology, Writing – review & editing. **Jacky Keung:** Conceptualization, Validation.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

Acknowledgments

This work was supported by the National Natural Science Foundation of China under Grants (No. 61972290), the Sanya Yazhou Bay Science and Technology City (No. SCKJ-JYRC-2022-17), the Youth Fund Project of Hainan Natural Science Foundation (No. 622QN344), and the Sanya Science and Education Innovation Park of Wuhan University of Technology (No. 2022KF0020).

References

Ahmad, W., Chakraborty, S., Ray, B., Chang, K.-W., 2020. A transformer-based approach for source code summarization. In: Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics. pp. 4998–5007.

Ahmad, W., Chakraborty, S., Ray, B., Chang, K.-W., 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. pp. 2655–2668.

Baltes, S., Dumani, L., Treude, C., Diehl, S., 2018. Sotorrent: Reconstructing and analyzing the evolution of stack overflow posts. In: Proceedings of the 15th International Conference on Mining Software Repositories. pp. 319–330.

Chatterjee, P., 2022. Automatic identification of informative code in stack overflow posts.

Chatterjee, P., Kong, M., Pollock, L., 2020. Finding help with programming errors: An exploratory study of novice software engineers' focus in stack overflow posts. J. Syst. Softw. 159, 110454.

Chen, Y., Dai, H., Yu, X., Hu, W., Xie, Z., Tan, C., 2021b. Improving ponzi scheme contract detection using multi-channel textCNN and transformer. Sensors 21 (19), 6417.

Chen, Y., Lu, X., 2020. Deep category-level and regularized hashing with global semantic similarity learning. IEEE Trans. Cybern. 51 (12), 6240–6252.

Chen, Y., Lu, X., Li, X., 2020a. Supervised deep hashing with a joint deep network. Pattern Recognit. 105, 107368.

Chen, Y., Lu, X., Wang, S., 2020b. Deep cross-modal image–voice retrieval in remote sensing. IEEE Trans. Geosci. Remote Sens. 58 (10), 7049–7061.

Chen, M., Tworek, J., Jun, H., Yuan, Q., Pinto, H.P.d.O., Kaplan, J., Edwards, H., Burda, Y., Joseph, N., Brockman, G., et al., 2021a. Evaluating large language models trained on code. arXiv preprint arXiv:2107.03374.

Chen, Y., Xiong, S., Mou, L., Zhu, X.X., 2022. Deep quadruple-based hashing for remote sensing image-sound retrieval. IEEE Trans. Geosci. Remote Sens. 60, 1–14.

Cheng, W., Hu, P., Wei, S., Mo, R., 2022. Keyword-guided abstractive code summarization via incorporating structural and contextual information. Inf. Softw. Technol. 150, 106987.

Chengran, Y., Xu, B., Thung, F., Shi, Y., Zhang, T., Yang, Z., Zhou, X., Shi, J., He, J., Han, D., et al., 2022. Answer summarization for technical queries: Benchmark and new approach. arXiv preprint arXiv:2209.10868.

Cobbe, K., Kosaraju, V., Bavarian, M., Hilton, J., Nakano, R., Hesse, C., Schulman, J., 2021. Training verifiers to solve math word problems. arXiv preprint arXiv:2110.14168.

Cohen, J., 1960. A coefficient of agreement for nominal scales. Educ. Psychol. Meas. 20 (1), 37–46.

Devine, P., Blincoe, K., 2022. Unsupervised extreme multi label classification of stack overflow posts. In: 2022 IEEE/ACM 1st International Workshop on Natural Language-Based Software Engineering (NLBSE). pp. 1–8.

Feng, Z., Guo, D., Tang, D., Duan, N., Feng, X., Gong, M., Shou, L., Qin, B., Liu, T., Jiang, D., et al., 2020. Codebert: A pre-trained model for programming and natural languages. In: Findings of the Association for Computational Linguistics: EMNLP 2020. pp. 1536–1547.

Feng, S., Keung, J., Yu, X., Xiao, Y., Zhang, M., 2021. Investigation on the stability of SMOTE-based oversampling techniques in software defect prediction. Inf. Softw. Technol. 139, 106662.

Fried, D., Aghajanyan, A., Lin, J., Wang, S., Wallace, E., Shi, F., Zhong, R., Yih, W.-t., Zettlemoyer, L., Lewis, M., 2022. Incoder: A generative model for code infilling and synthesis. arXiv preprint arXiv:2204.05999.

Gage, P., 1994. A new algorithm for data compression. C Users Journal 12 (2), 23–38.

Gao, Z., Xia, X., Grundy, J., Lo, D., Li, Y.-F., 2020. Generating question titles for stack overflow from mined code snippets. ACM Trans. Softw. Eng. Methodol. (TOSEM) 29 (4), 1–37.

Gao, Z., Xia, X., Lo, D., Grundy, J.C., Zhang, X., Xing, Z., 2022. I know what you are searching for: Code snippet recommendation from stack overflow posts. ACM Trans. Softw. Eng. Methodol. http://dx.doi.org/10.1145/3550150.

Guo, J., Liu, J., Wan, Y., Li, L., Zhou, P., 2022b. Modeling hierarchical syntax structure with triplet position for source code summarization. In: Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers). pp. 486–500.

Guo, D., Lu, S., Duan, N., Wang, Y., Zhou, M., Yin, J., 2022a. UniXcoder: Unified cross-modal pre-training for code representation. arXiv preprint arXiv:2203.03850.

He, C., Wu, J., Zhang, Q., 2020. Research leadership flow determinants and the role of proximity in research collaborations. J. Assoc. Inf. Sci. Technol. 71 (11), 1341–1356.

He, C., Wu, J., Zhang, Q., 2021. Characterizing research leadership on geographically weighted collaboration network. Scientometrics 126 (5), 4005–4037.

He, C., Wu, J., Zhang, Q., 2022a. Proximity-aware research leadership recommendation in research collaboration via deep neural networks. J. Assoc. Inf. Sci. Technol. 73 (1), 70–89.

He, J., Xu, B., Yang, Z., Han, D., Yang, C., Lo, D., 2022b. PTM4tag: Sharpening tag recommendation of stack overflow posts with pre-trained models. In: 2022 IEEE/ACM 30th International Conference on Program Comprehension (ICPC). pp. 1–11.

Hendrycks, D., Basart, S., Kadavath, S., Mazeika, M., Arora, A., Guo, E., Burns, C., Puranik, S., He, H., Song, D., et al., 2021. Measuring coding challenge competence with apps. arXiv preprint arXiv:2105.09938.

Hochreiter, S., Schmidhuber, J., 1997. Long short-term memory. Neural Comput. 9 (8), 1735–1780.

Holtzman, A., Buys, J., Du, L., Forbes, M., Choi, Y., 2019. The curious case of neural text degeneration. arXiv preprint arXiv:1904.09751.

Inala, J.P., Wang, C., Yang, M., Codas, A., Encarnación, M., Lahiri, S.K., Musuvathi, M., Gao, J., 2022. Fault-aware neural code rankers. arXiv preprint arXiv:2206.03865.

Kenton, J.D.M.-W.C., Toutanova, L.K., 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In: Proceedings of NAACL-HLT. pp. 4171–4186.

Khandelwal, U., He, H., Qi, P., Jurafsky, D., 2018. Sharp nearby, fuzzy far away: How neural language models use context. In: Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers). pp. 284–294.

Kincaid, J.P., Fishburne, Jr., R.P., Rogers, R.L., Chissom, B.S., 1975. Derivation of New Readability Formulas (Automated Readability Index, Fog Count and Flesch Reading Ease Formula) for Navy Enlisted Personnel. Technical Report, Naval Technical Training Command Millington TN Research Branch.

Kou, B., Di, Y., Chen, M., Zhang, T., 2022. Sosum: a dataset of stack overflow post summaries. In: Proceedings of the 19th International Conference on Mining Software Repositories. pp. 247–251.

Lewis, M., Liu, Y., Goyal, N., Ghazvininejad, M., Mohamed, A., Levy, O., Stoyanov, V., Zettlemoyer, L., 2020. BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In: Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics. pp. 7871–7880.

Li, Y., Choi, D., Chung, J., Kushman, N., Schrittwieser, J., Leblond, R., Eccles, T., Keeling, J., Gimeno, F., Lago, A.D., et al., 2022. Competition-level code generation with alphacode. arXiv preprint arXiv:2203.07814.

Li, Z., Wu, Y., Peng, B., Chen, X., Sun, Z., Liu, Y., Yu, D., 2021. Secnn: A semantic CNN parser for code comment generation. J. Syst. Softw. 181, 111036.

Lin, C.-Y., 2004. ROUGE: A package for automatic evaluation of summaries. In: Text Summarization Branches Out. Association for Computational Linguistics, pp. 74–81.

Lin, C.-Y., Och, F.J., 2004. Orange: a method for evaluating automatic evaluation metrics for machine translation. In: Proceedings of the 20th International Conference on Computational Linguistics. pp. 501–507.

Liu, K., Yang, G., Chen, X., Yu, C., 2022a. Sotitle: A transformer-based post title generation approach for stack overflow. arXiv preprint arXiv:2202.09789.

Liu, K., Yang, G., Chen, X., Zhou, Y., 2022b. EL-CodeBert: Better exploiting CodeBert to support source code-related classification tasks. In: Proceedings of the 13th Asia-Pacific Symposium on Internetware. pp. 147–155.

Liu, J., Zhou, P., Yang, Z., Liu, X., Grundy, J.C., 2018. FastTagRec: fast tag recommendation for software information sites. Autom. Softw. Eng. 25, 675–701.

Loshchilov, I., Hutter, F., 2017. Decoupled weight decay regularization. arXiv preprint arXiv:1711.05101.

Lu, S., Guo, D., Ren, S., Huang, J., Svyatkovskiy, A., Blanco, A., Clement, C., Drain, D., Jiang, D., Tang, D., et al., 2021. Codexglue: A machine learning benchmark dataset for code understanding and generation. arXiv preprint arXiv:2102.04664.

Ma, X., Keung, J., Yang, Z., Yu, X., Li, Y., Zhang, H., 2022. CASMS: Combining clustering with attention semantic model for identifying security bug reports. Inf. Softw. Technol. 147, 106906.

Mashhadi, E., Hemmati, H., 2021. Applying codebert for automated program repair of java simple bugs. In: 2021 IEEE/ACM 18th International Conference on Mining Software Repositories. MSR, IEEE, pp. 505–509.

Mondal, S., Saifullah, C.K., Bhattacharjee, A., Rahman, M.M., Roy, C.K., 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). pp. 1–11.

Nadi, S., Treude, C., 2020. Essential sentences for navigating stack overflow answers. In: 2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering. SANER, IEEE, pp. 229–239.

Nijkamp, E., Pang, B., Hayashi, H., Tu, L., Wang, H., Zhou, Y., Savarese, S., Xiong, C., 2022. A conversational paradigm for program synthesis. arXiv e-prints arXiv–2203.

Papineni, K., Roukos, S., Ward, T., Zhu, W.-J., 2002. Bleu: a method for automatic evaluation of machine translation. In: Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics. pp. 311–318.

Radford, A., Narasimhan, K., Salimans, T., Sutskever, I., Improving language understanding by generative pre-training.

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., Liu, P.J., 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. J. Mach. Learn. Res. 21, 1–67.

Robertson, S., Zaragoza, H., 2009. The probabilistic relevance framework: BM25 and beyond. Inf. Retr. 3 (4), 333–389.

Rubei, R., Sipio, C.D., Nguyen, P.T., Rocco, J.D., Ruscio, D.D., 2020. PostFinder: Mining stack overflow posts to support software developers. Inf. Softw. Technol. 127, 106367.

See, A., Liu, P.J., Manning, C.D., 2017. Get to the point: Summarization with pointer-generator networks. In: Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers). pp. 1073–1083.

Senter, R., Smith, E.A., 1967. Automated Readability Index. Technical Report, Cincinnati Univ OH.

Shi, F., Fried, D., Ghazvininejad, M., Zettlemoyer, L., Wang, S.I., 2022. Natural language to code translation with execution. arXiv preprint arXiv:2204.11454.

Shi, E., Wang, Y., Du, L., Zhang, H., Han, S., Zhang, D., Sun, H., 2021. Cast: Enhancing code summarization with hierarchical splitting and reconstruction of abstract syntax trees. arXiv preprint arXiv:2108.12987.

Tang, Z., Li, C., Ge, J., Shen, X., Zhu, Z., Luo, B., 2021. AST-transformer: Encoding abstract syntax trees efficiently for code summarization. In: 2021 36th IEEE/ACM International Conference on Automated Software Engineering. ASE, IEEE Computer Society, pp. 1193–1195.

Tang, Z., Shen, X., Li, C., Ge, J., Huang, L., Zhu, Z., Luo, B., 2022. AST-trans: Code summarization with efficient tree-structured attention. In: 2022 IEEE/ACM 44th International Conference on Software Engineering (ICSE). pp. 150–162.

Tu, Z., Lu, Z., Liu, Y., Liu, X., Li, H., 2016. Modeling coverage for neural machine translation. arXiv preprint arXiv:1601.04811.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł., Polosukhin, I., 2017. Attention is all you need. Adv. Neural Inf. Process. Syst. 30.

Wan, Y., Zhao, Z., Yang, M., Xu, G., Ying, H., Wu, J., Yu, P.S., 2018. Improving automatic source code summarization via deep reinforcement learning. In: Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering. pp. 397–407.

Wang, S., Lo, D., Vasilescu, B., Serebrenik, A., 2014. EnTagRec++: An enhanced tag recommendation system for software information sites. Empir. Softw. Eng. 23, 800–832.

Wang, Y., Wang, W., Joty, S., Hoi, S.C., 2021. Codet5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation. arXiv preprint arXiv:2109.00859.

Wang, X., Wei, J., Schuurmans, D., Le, Q., Chi, E., Zhou, D., 2022. Self-consistency improves chain of thought reasoning in language models. arXiv preprint arXiv:2203.11171.

Wang, X., Xia, X., Lo, D., 2015. TagCombine: Recommending tags to contents in software information sites. J. Comput. Sci. Tech. 30, 1017–1035.

Xia, X., Lo, D., Wang, X., Zhou, B., 2013. Tag recommendation in software information sites. In: 2013 10th Working Conference on Mining Software Repositories (MSR). pp. 287–296.

Xu, F.F., Alon, U., Neubig, G., Hellendoorn, V.J., 2022. A systematic evaluation of large language models of code. In: Proceedings of the 6th ACM SIGPLAN International Symposium on Machine Programming. pp. 1–10.

Xu, B., Hoang, T., Sharma, A., Yang, C., Xia, X., Lo, D., 2021. Post2Vec: Learning distributed representations of stack overflow posts. IEEE Trans. Softw. Eng. 1.

Xu, B., Xing, Z., Xia, X., Lo, D., 2017. AnswerBot: Automated generation of answer summary to developers' technical questions. In: 2017 32nd IEEE/ACM International Conference on Automated Software Engineering. ASE, IEEE, pp. 706–716.

Yang, Z., Keung, J., Kabir, M.A., Yu, X., Tang, Y., Zhang, M., Feng, S., 2021a. Acomnn: Attention enhanced compound neural network for financial time-series forecasting with cross-regional features. Appl. Soft Comput. 111, 107649.

Yang, Z., Keung, J., Yu, X., Gu, X., Wei, Z., Ma, X., Zhang, M., 2021b. A multi-modal transformer-based code summarization approach for smart contracts. In: 2021 IEEE/ACM 29th International Conference on Program Comprehension. ICPC, IEEE, pp. 1–12.

Yu, X., Bennin, K.E., Liu, J., Keung, J.W., Yin, X., Xu, Z., 2019a. An empirical study of learning to rank techniques for effort-aware defect prediction. In: 2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering. SANER, IEEE, pp. 298–309.

Yu, X., Keung, J., Xiao, Y., Feng, S., Li, F., Dai, H., 2022. Predicting the precise number of software defects: Are we there yet? Inf. Softw. Technol. 146, 106847.

Yu, X., Liu, J., Keung, J.W., Li, Q., Bennin, K.E., Xu, Z., Wang, J., Cui, X., 2019b. Improving ranking-oriented defect prediction using a cost-sensitive ranking SVM. IEEE Trans. Reliab. 69 (1), 139–153.

Yu, X., Liu, J., Yang, Z., Liu, X., 2017. The Bayesian network based program dependence graph and its application to fault localization. J. Syst. Softw. 134, 44–53.

Yu, X., Wu, M., Jian, Y., Bennin, K.E., Fu, M., Ma, C., 2018. Cross-company defect prediction via semi-supervised clustering-based data filtering and MSTrA-based transfer learning. Soft Comput. 22 (10), 3461–3472.

Zhang, A., Fang, L., Ge, C., Li, P., Liu, Z., 2022a. Efficient transformer with code token learner for code clone detection. J. Syst. Softw. 111557.

Zhang, F., Keung, J., Yu, X., Xie, Z., Yang, Z., Ma, C., Zhang, Z., 2022b. Improving stack overflow question title generation with copying enhanced codeBERT model and bi-modal information. Inf. Softw. Technol. 106922.

Zhao, K., Liu, J., Xu, Z., Liu, X., Xue, L., Xie, Z., Zhou, Y., Wang, X., 2022. Graph4Web: A relation-aware graph attention network for web service classification. J. Syst. Softw. 190, 111324.

Zhao, K., Xu, Z., Yan, M., Zhang, T., Xue, L., Fan, M., Keung, J., 2023. The impact of class imbalance techniques on crashing fault residence prediction models. Empirical Software Engineering 28 (2), 49.

Zhen, Y., KEUNG, J.W., Xiao, Y., Yan, X., Zhi, J., ZHANG, J., 2022. On the significance of category prediction for code-comment synchronization. ACM Trans. Softw. Eng. Methodol..

Zhou, P., Liu, J., Liu, X., Yang, Z., Grundy, J., 2019. Is deep learning better than traditional approaches in tag recommendation for software information sites? Inf. Softw. Technol. 109, 1–13.

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

Zhou, Y., Shen, J., Zhang, X., Yang, W., Han, T., Chen, T., 2022a. Automatic source code summarization with graph attention networks. J. Syst. Softw. 188, 111257.

Zhou, Z., Yu, H., Fan, G., Huang, Z., Yang, X., 2022b. Summarizing source code with hierarchical code representation. Inf. Softw. Technol. 143, 106761.

Fengji Zhang is currently a master student at the School of Computer Science, Wuhan University, China. He also received the B.S. degree from the School of Computer Science, Wuhan University in 2020. His current research interests include intelligent software engineering and natural language processing.

Jin Liu received the Ph.D. degree in computer science from the State Key Lab of Software Engineering, Wuhan University in 2005. He is currently a professor at School of Computer Science, Wuhan University. He has authored or coauthored a number of research papers in international journals. His research interests include mining software repositories and intelligent information processing.

Yao Wan is currently a lecturer with the College of Computer Science and Technology at Huazhong University of Science and Technology, Wuhan, China. Prior to that, He received the Ph.D. degree from Zhejiang University, China, in 2019. His current research focuses on the synergy between artificial intelligence and software engineering, especially on natural language processing, programming language analysis, data mining and machine learning.

Xiao Yu received the Ph.D. degree from the Department of Computer Science, City University of Hong Kong in 2021. He is currently an assistant professor at School of Computer Science and Technology, Wuhan University of Technology. His research interests include software engineering and deep learning.

Xiao Liu received his Ph.D. degree in computer science and software engineering from the Faculty of Information and Communication Technologies, Swinburne University of Technology, Melbourne, Australia, in 2011. He is currently an associate professor with the School of Information Technology, Deakin University, Melbourne. His current research interests include workflow systems, cloud and edge computing, big data analytics, and human-centric software engineering.

Jacky Keung received the Ph.D. degree in software engineering from the University of New South Wales, Australia. He is currently an associate professor with the Department of Computer Science, City University of Hong Kong, Hong Kong, China. His current research interests include software engineering, data mining, and deep learning.