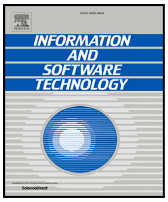
Information and Software Technology 148 (2022) 106922

Contents lists available at ScienceDirect

Information and Software Technology

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



Improving Stack Overflow question title generation with copying enhanced CodeBERT model and bi-modal information

Fengji Zhang [a,b], Xiao Yu [a,c,d,*], Jacky Keung [e], Fuyang Li [a,c,*], Zhiwen Xie [b], Zhen Yang [e], Caoyuan Ma [b], Zhimin Zhang [b]

[a] *School of Computer Science and Artificial Intelligence, Wuhan University of Technology, Wuhan, China*
[b] *School of Computer Science, Wuhan University, Wuhan, China*
[c] *Wuhan University of Technology Chongqing Research Institute, Chongqing, China*
[d] *Sanya Science and Education Innovation Park of Wuhan University of Technology, Sanya, China*
[e] *Department of Computer Science, City University of Hong Kong, Hong Kong, China*

ARTICLE INFO

Keywords:
Stack Overflow
Title generation
Copy mechanism
CodeBERT

ABSTRACT

Context: Stack Overflow is very helpful for software developers who are seeking answers to programming problems. Previous studies have shown that a growing number of questions are of low quality and thus obtain less attention from potential answerers. Gao et al. proposed an LSTM-based model (i.e., BiLSTM-CC) to automatically generate question titles from the code snippets to improve the question quality. However, only using the code snippets in the question body cannot provide sufficient information for title generation, and LSTMs cannot capture the long-range dependencies between tokens.
Objective: This paper proposes CCBERT, a deep learning based novel model to enhance the performance of question title generation by making full use of the bi-modal information of the entire question body.
Method: CCBERT follows the encoder–decoder paradigm and uses CodeBERT to encode the question body into hidden representations, a stacked Transformer decoder to generate predicted tokens, and an additional copy attention layer to refine the output distribution. Both the encoder and decoder perform the multi-head self-attention operation to better capture the long-range dependencies. This paper builds a dataset containing around 200,000 high-quality questions filtered from the data officially published by Stack Overflow to verify the effectiveness of the CCBERT model.
Results: CCBERT outperforms all the baseline models on the dataset. Experiments on both code-only and low-resource datasets show the superiority of CCBERT with less performance degradation. The human evaluation also shows the excellent performance of CCBERT concerning both readability and correlation criteria.
Conclusion: CCBERT is capable of automatically capturing the bi-modal semantic information from the entire question body and parsing the long-range dependencies to achieve better performance. Therefore, CCBERT is an effective approach for generating Stack Overflow question titles.

1. Introduction

Stack Overflow (SO) is one of the most thriving communities where software developers can seek answers to programming problems from peers. The open-data policy of SO has been attracting intense research interests [1–6]. The recent study of Mondal et al. [4] shows that a growing number of open questions in SO remain unanswered, partly because some developers fail to write high-quality questions. The SO community has given many practical writing suggestions in the official tutorial[1] to tackle this problem, and researchers have also made great efforts to help improve question quality in many ways [7–9].

Previous studies [10–13] in SO have demonstrated the importance of question titles to the overall quality of questions. Recently, Gao et al. [9] for the first time proposed an approach of automatically generating question titles from given code snippets. They used the BiLSTM-CC model, which is a Bi-directional Long Short-Term Memory network incorporated with the Copy [14] and Coverage [15] mechanism to generate titles from code snippets mined in corresponding

[*] Corresponding authors at: School of Computer Science and Artificial Intelligence, Wuhan University of Technology, Wuhan, China.
E-mail addresses: zhangfengji@whu.edu.cn (F. Zhang), xiaoyu@whut.edu.cn (X. Yu), jacky.keung@cityu.edu.hk (J. Keung), fyli@whut.edu.cn (F. Li), xiezhiwen@whu.edu.cn (Z. Xie), zhyang8-c@my.cityu.edu.hk (Z. Yang), macaoyuan@whu.edu.cn (C. Ma), zhangzhimin@whu.edu.cn (Z. Zhang).

[1] https://stackoverflow.com/help/how-to-ask.

https://doi.org/10.1016/j.infsof.2022.106922
Received 25 August 2021; Received in revised form 16 March 2022; Accepted 29 March 2022
Available online 6 April 2022

question bodies. Despite the encouraging performance, we argue that LSTMs may lack the ability to parse long-range dependencies according to Khandelwal et al. [16]. In addition, developers are not recommended to write only source code as questions in the SO community. Since a question body usually consists of the bi-modal content (i.e., text descriptions and code snippets), the information that developers infer from code snippets without surrounding contexts can be broken and misleading.

In this paper, we redefine the task proposed by Gao et al. [9] to **T**itle **G**eneration from the **E**ntire **Q**uestion **B**ody, namely **TGEQB**. We formulate this task as an abstractive summarization problem, and also propose our **CCBERT** model, which combines the **C**opy mechanism [14] to handle rare tokens and the pre-trained **C**ode**BERT** [17] model to parse bi-modal content. We follow the encoder–decoder paradigm and use CodeBERT to encode question bodies into hidden representations, a stacked Transformer decoder to generate predicted tokens, and an additional copy attention layer to refine the output distribution. Our encoder and decoder perform the multi-head self-attention operation, which helps CCBERT better capture the long-range dependencies than LSTMs.

To verify the effectiveness of our model, we conduct the empirical study by raising the following Research Questions (RQs):

RQ-1 Does our CCBERT model outperform the baseline models? We build a large-scale dataset Data$_{\text{exp}}$ with around 200,000 high-quality questions filtered from the data[2] officially published by Stack Overflow in December 2020, which contains all the historical questions from 2008 to 2020. We employ BLEU and ROUGE as the evaluation metrics and choose four baseline models (i.e., TF–IDF [18], BiLSTM [19], BiLSTM-CC [9], and BART [20]). Experimental results show that CCBERT outperforms all the baseline models regarding all the metrics.

RQ-2 What is the advantage of using the bi-modal information of the entire question body? We build a code-only dataset and choose BiLSTM-CC to compare with our model. Experimental results show that applying bi-modal information greatly boosts both models' performance, where CCBERT still outperforms BiLSTM-CC.

RQ-3 How effective is our CCBERT model under low-resource circumstances? We build three new train sets sized of 98,909 (Data$_{\text{exp}}$/2), 49,454 (Data$_{\text{exp}}$/4), and 24,727 (Data$_{\text{exp}}$/8) to train the CCBERT and BiLSTM-CC models. According to the experimental results, our CCBERT model shows significant superiority under low-resource circumstances compared with BiLSTM-CC.

RQ-4 How much influence does interrogative constraint have on model training? We follow Gao et al. [9] to apply the interrogative constraint when building Data$_{\text{exp}}$, which may make it easier to train our models. However, our model can be biased because of this. We build another dataset $Data_{exp+}$ to quantify the influence, and results show that dropping the interrogative constraint will lead to a decline in the performance of both CCBERT and BiLSTM-CC models.

RQ-5 How effective is our CCBERT model under human evaluation? Automated evaluation is not always trustworthy. So, we perform a more human-centered evaluation to investigate the overall quality of the titles generated by our models. Results show that our CCBERT performs much better than TF–IDF and BiLSTM-CC concerning the correlation criteria but a little bit worse than human written titles retrieved by TF–IDF concerning the readability criteria under human evaluation.

The contributions of this paper are as follows:

- We introduce a new task named TGEQB, which is to generate high-quality titles from the entire question body containing bi-modal content to help improve the question quality.

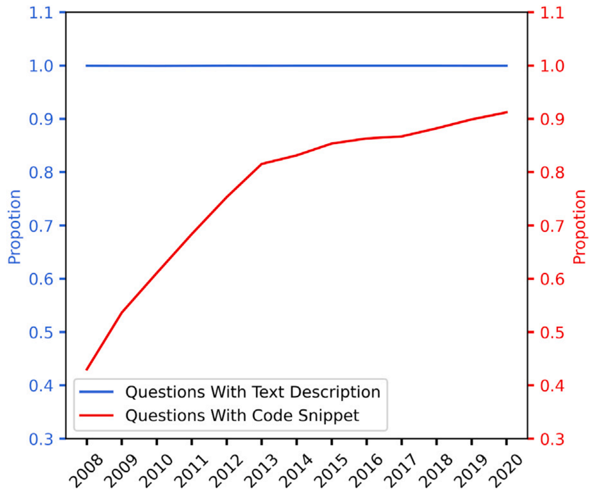


Fig. 1. The proportion of questions with text descriptions or code snippets.

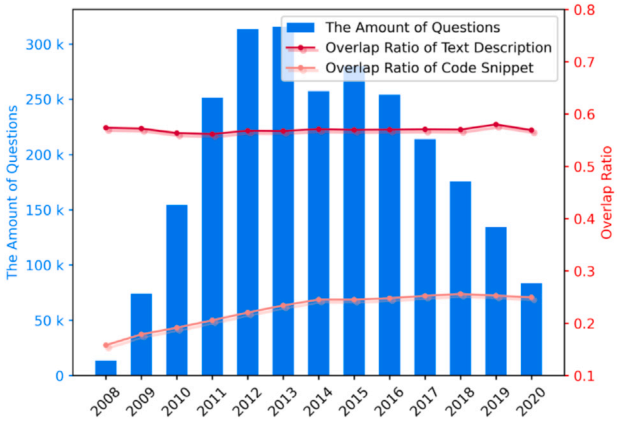


Fig. 2. The overlap between titles and code snippets/text descriptions.

- We propose a novel model named CCBERT, which combines the copy mechanism and CodeBERT to handle rare tokens and long-range dependencies in the bi-modal context.
- We have released our dataset and all relevant source code[3] to facilitate future research and application.

We organize the rest of this paper as follows: Section 2 reveals the motivation of our work. Section 3 introduces the details of our proposed approach. Section 4 describes the basic setup of our experiment, including the baseline models, evaluation metrics, and model settings. Section 5 presents the experimental results. 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. Motivation

We share similar user scenarios with Gao et al. [9], where less experienced developers or non-native English speakers may not be able to adequately describe their questions according to the writing rules suggested by SO. We can use an automated data-driven approach to help developers draft high-quality question titles in such circumstances. However, we argue that one should consider both text descriptions

[2] https://archive.org/download/stackexchange.

[3] https://github.com/zfj1998/SO_Title_Generation.

and code snippets when writing question titles. We also have concerns about the long-range dependency issues of using the entire question body as input. Therefore, this section aims to investigate the necessity of title generation using bi-modal content and the challenge of long sequence parsing.

2.1. Importance of bi-modal content

Our first step is to get a collection of high-quality samples for statistics. We believe that high-quality question posts should be clear and complete so that developers in the SO community are more likely and willing to answer. But it is a non-trivial task to evaluate the clearness and completeness of a post automatically. Therefore, we first filter the question posts based on the feedback they received from the SO community:

1. The question is not closed;
2. The question has an accepted answer;
3. The question gets more than one vote.

After filtering out the ones that do not meet the above feedback-related constraints, we obtain a collection of 3.2 million question posts that we regard as candidates of high-quality ones.

Intuitively, a programming question should always contain both text and code. Besides, when drafting a new question post in SO, the website will give the three suggestions:[4]

1. Summarize the problem;
2. Describe what you have tried;
3. Show some code.

We separately count the high-quality candidate questions containing text descriptions and code snippets by year. To be specific, question posts in the source file[5] are organized in a unified HTML format, so we extract the content wrapped by "<code></code>" tags as code snippets and the rest as text descriptions. We draw a line chart in Fig. 1 to show the statistical results, where the x-axis denotes the year and the y-axis denotes the proportion of questions with text descriptions or code snippets. We find that the proportion of high-quality candidates containing text descriptions is almost unchanging (100%) every year. While the proportion of high-quality candidates containing code snippets has been increasing in recent years, reaching 90% in 2020.

In addition, we have manually studied a number of high-quality candidates posted recently without code snippets. We find that many of them are not programming-related questions, such as software/ platform instructions,[6] knowledge Q&A,[7] etc. Some others may put the code in images[8] or external links.[9] The above-mentioned question posts are beyond the scope of this preliminary study.

Therefore, we believe that containing the content of both modalities helps improve the clearness and completeness of programming questions in SO. In our study, only the question posts containing bi-modal content and meeting the three feedback-related constraints are considered high-quality.

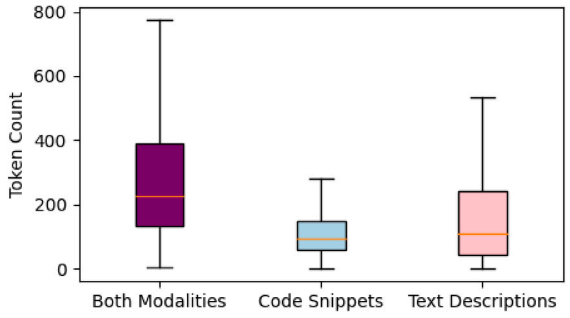


Fig. 3. The length distribution of code snippets and text descriptions in the question body.

2.2. Impact of bi-modal content on titles

The title of a question post is always dependent on the overall semantics of the question body. However, it is a non-trivial task to tell the precise impact of the bi-modal content in the question body on writing the title. In this subsection, we conduct a lexical-level statistical experiment and a human evaluation to estimate such impact.

First, we extract the high-quality question posts from 2008 to 2020, which contain bi-modal content and meet the three feedback-related constraints. Then we count the tokens that appear both in the title and text descriptions/code snippets, and draw a line chart in Fig. 2 to demonstrate the average overlap ratios by year. Moreover, we combine Fig. 2 with a bar chart to demonstrate the number of statistical samples per year. Specifically, the x-axis denotes the year, the left y-axis denotes the amount of questions, and the right y-axis denotes the overlap ratios between titles and text descriptions/code snippets.

From Fig. 2, we may find that the overlap ratios of high-quality questions have been stable since 2014. This may indicate that a certain extent of token overlap between the title and the bi-modal content ensures the title quality. To investigate this issue, we further perform a manual analysis.

We consider two criteria when manually evaluating post titles, either of which can be scored between 1 and 4. We illustrate the detailed descriptions and scoring standards in Table 1. We split the filtered high-quality posts into four categories according to their average overlap ratios between titles and the bi-modal content. And then, we randomly sample 500 question posts from each category. Five independent graduate students who are experienced programmers and familiar with Stack Overflow are invited to rate the titles based on the scoring standards, and each participant is assigned 100 posts per category. From the evaluation results in Table 2, we can find both the readability and correlation scores degrade when the overlap ratio is too low or too high. According to our participants, when the overlap is too low, the title tends to be vague. When the overlap is too high, the title tends to be the bare error report of a program. Both situations require more effort for the reader to fully understand the question. This suggests that properly borrowing tokens from the bi-modal content makes the title more expressive and matching the points of the question. Therefore, we have reasons to believe that bi-modal content is essential to writing good titles.

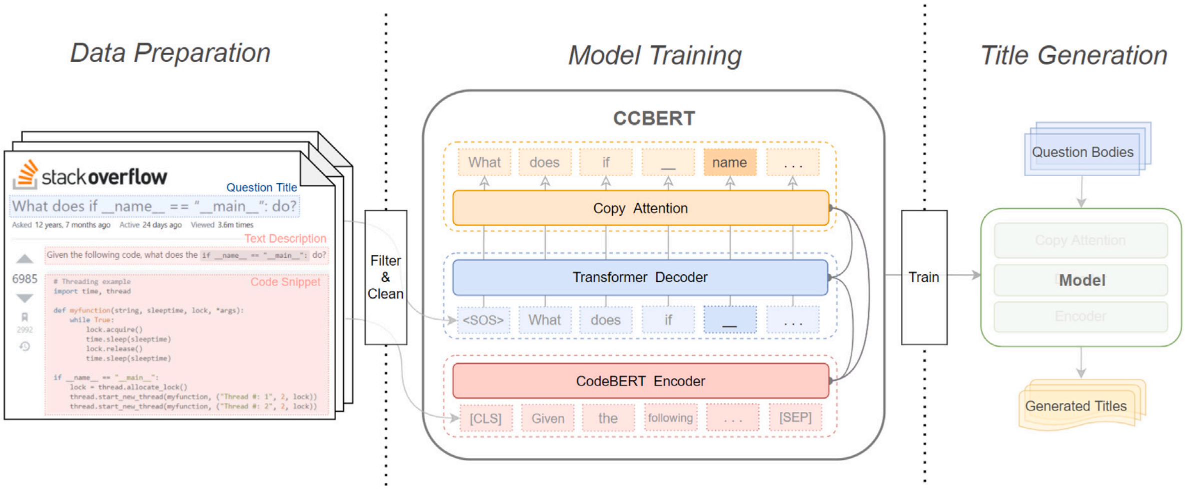
2.3. The long-range dependency issue

We draw a box plot in Fig. 3 to represent the length distributions of the entire question body, the code snippets, and the text descriptions of the high-quality question posts extracted in Section 2.2. We can find that the code snippets only occupy less than half of the body content, and the entire content of a question body can be very long, bringing new challenges to our title generation models. Specifically, over 56% questions have more than 200 tokens in their bodies, and

[4] https://stackoverflow.com/questions/ask.
[5] The *stackoverflow.com-Posts.7z* file.
[6] https://stackoverflow.com/questions/59553413/firebase-storage-image-not-showing.
[7] https://stackoverflow.com/questions/59554837/uml-how-to-model-either-or-both-union-concept.
[8] https://stackoverflow.com/questions/59552547/ios-swiftui-how-to-bring-up-extra-actions-like-embed-in-vstack-when-interactin.
[9] https://stackoverflow.com/questions/59552571/how-to-check-if-fixed-width-integers-are-defined.

Table 1
The criteria used for manually evaluating the titles. The evaluation score of each criteria is between 1 and 4.

Criteria	Description	Title scoring standard
Readability	Ignoring the content, considering the grammaticality and fluency of the title	1. Has too many errors to read and understand 2. Has minor errors but is readable and understandable 3. Is very easy to read and understand 4. Is very expressive and appealing
Correlation	Considering the consistency between the question and the title	1. Is totally missing the point of the question 2. Is relevant to the main point of the question 3. Is a good match of the question's points 4. Is a perfect summary of the question



What does if __name__ == "__main__": do?

Asked 12 years, 7 months ago Active 24 days ago Viewed 3.6m times

Given the following code, what does the `if __name__ == "__main__":` do?

```
# Threading example
import time, thread

def myfunction(string, sleeptime, lock, *args):
    while True:
        lock.acquire()
        time.sleep(sleeptime)
        lock.release()
        time.sleep(sleeptime)

if __name__ == "__main__":
    lock = thread.allocate_lock()
    thread.start_new_thread(myfunction, ("Thread #: 1", 2, lock))
    thread.start_new_thread(myfunction, ("Thread #: 2", 2, lock))
```

Fig. 4. The framework of our approach for Stack Overflow question title generation.

Table 2
Human evaluation results of the title quality with different overlap ratios with the bi-modal content. The ratios of four different scores and the average score are listed grouped by different criteria.

Criteria	Overlap ratio	Score 1	Score 2	Score 3	Score 4	Avg score
Readability	0–0.2	–	21.6%	77.4%	1.0%	2.794
	0.2–0.4	–	7.8%	88%	4.2%	2.964
	0.4–0.6	–	11%	85.6%	3.4%	2.924
	0.6–1.0	–	26%	73.8%	0.2%	2.742
Correlation	0–0.2	–	11.2%	87%	1.8%	2.906
	0.2–0.4	–	2.8%	86.4%	10.8%	3.080
	0.4–0.6	–	2.8%	89%	8.2%	3.054
	0.6–1.0	–	6.2%	90.6%	3.2%	2.970

some questions even have 500 tokens and more. The LSTM structure is only capable of using 200 tokens of context on average according to Khandelwal et al. [16], so we apply Transformer-based pre-trained models to tackle this problem. Later in Section 5, we will make further comparisons of these two models.

3. Proposed approach

We aim to help developers write high-quality questions with a better chance to get answers in Stack Overflow. Considering that using only code snippets[12] is not enough to generate high-quality titles, we introduce a new title generation task named TGEQB, which utilizes the bi-modal information of both text descriptions and code snippets in the question body. Following the general practice in machine learning studies, the framework of our approach demonstrated in Fig. 4 contains three main steps: data preparation, model training, and validation. We describe the details of our approach in this section, including the data preparation procedures and the detailed architecture of our CCBERT model.

3.1. Data preparation

Further filtering, tokenization, and partitioning are performed on the high-quality question posts extracted in Section 2.2 before we finally get the experimental dataset $Data_{exp}$.

Firstly, considering the vocabulary of our CodeBERT encoder was built on a dataset[10] concerning only six programming languages: *Java*, *Python*, *JS(JavaScript)*, *PHP*, *Ruby*, and *Go*, we also focus on the questions tagged with these programming languages in this preliminary study. To avoid the influence of noise data, we further filter out posts tagged with other popular languages in SO, including *C#*, *HTML*, and *C++*. In the end, we find the amounts of filtered *Ruby* and *Go* questions[11] far from enough for training and testing, which leaves us the questions tagged with only four programming languages: *Java*, *Python*, *JS*, and *PHP*.

Secondly, we notice that in Gao et al.'s work [9], they only kept the questions containing interrogative keywords: *how*, *what*, *why*, *which*, and *when* in their titles. While only 1/3 of our filtered high-quality question posts satisfy this constraint. After manually examining our data samples, we find that question titles without interrogatives tend to be more casual and are always incomplete sentences, which undoubtedly brings many difficulties for the model training. So in this preliminary study, we choose to apply this constraint in our primary dataset $Data_{exp}$ and perform an additional experiment later in Section 5.4 to investigate the performance of our models without this constraint.

In addition, we notice that the NLTK tokenizer[12] cannot separate special tokens in code snippets well, which leads to an extensive vocabulary and exacerbates the out-of-vocabulary issue. So we choose

[10] CodeSearchNet https://github.com/github/CodeSearchNet.

[11] There are only approximately 7000 and 3500 filtered questions for *Ruby* and *Go*.

[12] http://www.nltk.org/api/nltk.tokenize.html.

Table 3
The partition size of $Data_{exp}$.

Language	Train	Validation	Test
Java	57,118	2000	2000
Python	60,458	2000	2000
JS	53,708	2000	2000
PHP	26,535	1000	1000
Total	197,819	7000	7000

a simple tokenizing algorithm to tackle this problem. Specifically, there are three kinds of printable characters in the ASCII charset, including the digits (*0 to 9*), letters (*A/a to Z/z*), and punctuation symbols. We first put a white space on both sides of punctuation symbols in a string during tokenization and then split the string into tokens by white spaces. This way, we get a smaller vocabulary but longer sequences in return. Handling very long sequences is still an open problem in the field of deep learning [21–23]. In this preliminary study, we choose to filter out 5% of the question posts whose body length exceeds 1000, or the title length exceeds 25.

As for data partitioning, we sort the questions in chronological order and choose the latest samples for testing and the rest for training. Because we think it is more applicable to the real-world application if the models take past questions for training and new ones for testing. Besides, we believe our time-wise partitioning will help relieve the target leakage problem caused by the homogeneous questions between the train and test sets. We show the statistics of our processed dataset in Table 3.

3.2. The CCBERT model

We propose CCBERT, a novel model combining the pre-trained CodeBERT model and the copy mechanism. It is an attentional encoder–decoder system, which can be trained and used in an end-to-end manner. Our model architecture is illustrated in Fig. 5. Specifically, we apply the CodeBERT model and Transformer-decoder layers in their original form and put a specialized copy attention layer above the encoder and decoder. Formally, given a token sequence $X = [x_1, x_2, \dots, x_n]$ of a question body and a token sequence $T = [t_1, t_2, \dots, t_l]$ of its corresponding title sampled from our dataset, CCBERT learns to generate T based on X.

3.2.1. CodeBERT encoder

Unlike the general models used for summarization [24–26], our model needs to understand both Natural Language (NL) and Programming Language (PL), which is determined by the characteristics of our dataset. Recently, Feng et al. [17] introduced CodeBERT, which was pre-trained on a vast scale dataset extracted from Github repositories containing source code and code comments. This way, CodeBERT can capture the semantic relationship between NL and PL, and produce vector representations that support downstream tasks, such as defect prediction [27–31], program repair [32], etc.

We use the pre-trained CodeBERT as our encoder. It is a stack of multiple Transformer-encoder layers which mainly performs bidirectional self-attention operations. Formally, given a question body containing text descriptions and code snippets, we first turn it into a sequence of tokens with a byte pair encoding tokenizer built in the CodeBERT model. Then, we surround the token sequence with two special tokens[13] to be consistent with the data format used during pre-training and get the final input sequence X,

$$X = \left[x_{CLS}, x_1, x_2, \dots, x_n, x_{SEP}\right]. \tag{1}$$

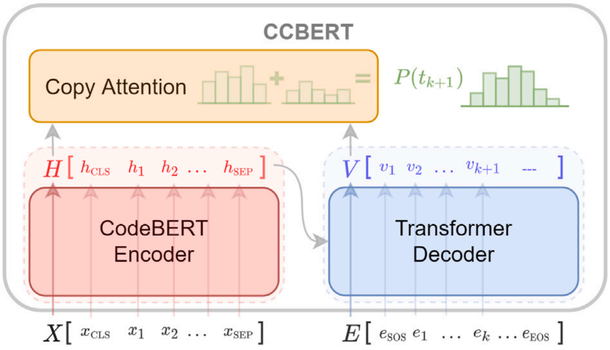

CCBERT
Copy Attention
+
=
$P(t_{k+1})$
H [h_{CLS} h_1 h_2 … h_{SEP}]
V [v_1 v_2 … v_{k+1} ---]
CodeBERT Encoder
Transformer Decoder
X [x_{CLS} x_1 x_2 … x_{SEP}]
E [e_{SOS} e_1 … e_k … e_{EOS}]

Fig. 5. The detailed structure of CCBERT at the $(k+1)$th decoding step.

After that, we feed X to the encoder and get a matrix H that consists of the encoded vectors of all input tokens

$$H = \text{ENCODER}(X), \tag{2}$$

where $H = [h_{CLS}, h_1, h_2, \dots, h_n, h_{SEP}]$ and each vector h_i is a hidden representation of the semantic relationship of a token against others.

3.2.2. Transformer decoder

After encoding the input question body, we need the decoder to generate the hidden representation of each token in the predicted question title. Since the nature of the CodeBERT encoder is Transformer-encoder layers, we stack several layers of vanilla Transformer-decoder [33] as our decoder.

Formally, suppose we have generated the first k tokens (i.e., $Y = \left[y_{SOS}, y_1, y_2, \dots, y_k\right]$)[14] of the predicted title, and now are going to generate the $(k+1)$th token (i.e, at the $(k+1)$th decoding step). We first use the same embedding layer of the encoder to turn the input sequence Y into a matrix E containing the embedding vectors of tokens (i.e, $E = \left[e_{SOS}, e_1, e_2, \dots, e_k\right]$). The input for the decoder is two-fold, one is the hidden vectors H provided by the encoder, the other is the embedding E of generated sequence. We feed H and E to the decoder and get a matrix V containing the hidden representations of $k+1$ predicted tokens

$$V = \text{DECODER}(H, E), \tag{3}$$

where $V = [v_1, v_2, \dots, v_{k+1}]$. We take v_{k+1} as the hidden representation of the $(k+1)$th predicted token.

3.2.3. Copy attention layer

Usually, we can have several linear layers above the decoder to map the hidden representation v_{k+1} to its most likely token in the vocabulary. However, according to the statistics, question titles have a high overlap with the body content. Besides, we should also pay attention to some essential but rare tokens, such as variable names, class libraries, application frameworks, etc. In this case, we incorporate the copy mechanism to facilitate our model to copy tokens directly from the body content when generating titles. The copy mechanism was first introduced in the pointer-generator network [26], which was originally applied to the Recurrent Neural Networks (RNNs). In our work, we implement a specialized copy attention layer to adapt the copy mechanism to our Transformer-based model.

Formally, when generating the $(k+1)$th token in the predicted title, we first need to calculate the attention vector a_{k+1} with the encoder hidden state H, the embedding vector of the generated token e_k, and the hidden representation v_{k+1} of the $(k+1)$th predicted token,

$$a_{k+1} = Attention(H, e_k, v_{k+1}). \tag{4}$$

[13] The *SEP* token marks the end of a sentence. The *CLS* token is put in front of the input sequence and specially used for sentence classification.

[14] *SOS* means the start of a sequence.

Then, we use the attention vector a_{k+1} and the encoder hidden state matrix H to get a single vector $context_{k+1}$ as the overall "context" of the input sequence,

$$context_{k+1}^{\mathsf{T}} = a_{k+1}^{\mathsf{T}} \cdot H, \tag{5}$$

where T represents the transpose symbol. After that, with the input context $context_{k+1}$ and the current decoder state v_{k+1}, we can get the probability distribution P_{vocab} of each token in the vocabulary to be chosen as the $(k+1)$th predicted token,

$$P_{vocab} = LinearSoftmax(context_{k+1}, v_{k+1}), \tag{6}$$

where $LinearSoftmax$ is a linear neural network with the Softmax output layer.

Usually, we can choose the token with the highest probability in P_{vocab} as the predicted token. But to incorporate the copy mechanism, we have to calculate an additional probability p_{copy} as a soft switch to choose between generating a word from the vocabulary by sampling from P_{vocab}, or copying a word from the input sequence by sampling from the attention distribution a_{k+1},

$$p_{copy} = LinearSigmoid(context_{k+1}, v_{k+1}, e_k), \tag{7}$$

where $LinearSigmoid$ is a linear layer with the Sigmoid activation function. Finally, we can get the revised probability distribution $P(t_{k+1})$ of choosing the $(k+1)$th token,

$$P(t_{k+1}) = p_{copy} \sum_{i:x_i=t_{k+1}}^{n} a_{(k+1)_i} + (1 - p_{copy}) P_{vocab}(t_{k+1}). \tag{8}$$

We generate each token recursively and stop when the EOS[15] token comes up. The overall trainable parameters θ include those of the stacked Transformer-encoder layers in CodeBERT, the stacked Transformer-decoder layers in our decoder, and the linear neural networks in our copy attention layer. The training loss is the negative log-likelihood of each token in the target sequence, which we use to update θ through backpropagation to maximize the likelihood between the generated titles and the original ones in our dataset during training.

4. Experimental setup

This section illustrates the baseline models, the evaluation metrics, and the hyperparameter settings for our CCBERT model.

4.1. Comparisons

To demonstrate how competitive CCBERT is, we choose several state-of-the-art models as baselines, which have been widely studied in the field of natural language processing. We briefly introduce the general ideas of these models in the following.

(1) **TF–IDF** This method is a classic full text searching algorithm, its name stands for "Term Frequency (TF) × Inverse Document Frequency (IDF)". TF–IDF is a weighting algorithm for a bag-of-words language model. Specifically, the "bag" contains a list of unique terms sourced from a given corpus. A paragraph can be turned into a vector by counting its in-bag terms' frequency (TF). Because the probability of a term's occurrence is often in inverse proportion to its importance, one can use the term frequency of appearing in all documents (IDF^{-1}) to divide TF and get the revised weight of each term. This way, we can calculate the distance between paragraphs in the vector space. In our experiment, we use Lucene[16] to find the most similar question in the train set given a question body.

(2) **BiLSTM** Long Short Term Memory networks (LSTMs) are a special kind of RNNs, with an additional cell state and three carefully designed "gates" to alleviate the problem of long-term dependencies. The idea of Bidirectional LSTMs (BiLSTMs) is to duplicate the first recurrent layer in the network and then provide the input sequence to the first layer and a reversed copy of the input to the second. This way, all available information in the past and future of a specific processing step can be considered during training. We stack two BiLSTM layers as the encoder and two LSTM layers as the decoder, along with the attention mechanism introduced by Bahdanau et al. [34] to build a model as our baseline, which we refer to as BiLSTM.

(3) **BiLSTM-CC** This was the method used by Gao et al. [9] to generate question titles from code snippets. It shares the same structure as the BiLSTM model mentioned above, except to assemble another two non-trivial mechanisms. One is the copy mechanism we have illustrated above; the other is the coverage mechanism. Tu et al. [15] first introduced the "coverage" vector that keeps track of the attention history and further facilitates the attention calculation so that a neural machine translation system would consider more about untranslated words. Gao et al. [9] took advantage of the coverage penalty to suppress meaningless repetitions during generation. In our experiment, we build the BiLSTM and BiLSTM-CC models with OpenNMT,[17] which is a well-acknowledged framework to build sequence-to-sequence models.

(4) **BART** Lewis et al. [20] proposed the BART model to bridge the gap between pre-trained Bidirectional encoder (i.e. BERT [35]) and pre-trained Auto-Regressive Transformer (i.e. GPT [36]), which are good at comprehension and generation tasks respectively. BART is pre-trained under the supervision of several denoising objectives, where input text is corrupted by a stochastic noising function and the model is demanded to reconstruct the original text. BART is particularly effective when fine-tuned for neural machine translation and abstractive text summarization tasks, such as WMT, CNN/DailyMail, XSum etc. We use the open-source code and pre-trained parameters[18] for BART to validate its performance on our dataset.

In addition to the above baselines, we implement an oracle method to show the best performance of an extractive model.

(5) **Oracle** The idea of extractive summarization, which is to select primary sentences that best match the target summary, inspires us to explore the possibility of making up a title only using tokens that appear in the question body. However, there are millions of permutations of a title containing tens of tokens, which is more complicated than selecting and arranging sentences. In addition, tokens arranged in the correct order do not necessarily make sense. So, instead of building another baseline model, we remove the tokens in a question title if they are not in the question body and keep the rest as the "generated" title to simulate the best performance of an extractive model. Considering our objective is to maximize the BLEU and ROUGE score, we follow the work of Liu et al. [25] and implement another method based on beam search (with 20 beam width) to find the permutation that performs the best on these two metrics. It turns out that the second method does no better than the first one on both metrics due to the limited searching space. Therefore, we use the simple method mentioned above as the oracle method indicating the best possible result from a model.

[15] EOS means the end of a sequence.

[16] Apache Lucene computes the similarity using TF–IDF by default.

[17] https://opennmt.net.

[18] https://huggingface.co/facebook/bart-base.

4.2. Automated evaluation metrics

Since the nature of our task is a sequence generation problem, where BLEU and ROUGE are the most commonly used metrics, we choose both to measure the precision- and recall-oriented similarity between the generated titles and the original ones.

4.2.1. BLEUS-4

The Bi-Lingual Evaluation Understudy (BLEU) method was first introduced by Papineni et al. [37] to measure the performance of a translation system. Given the candidate translations and reference sentences, the first step in this method is to compute the *ngram* precision,

$$p_n = \frac{\sum_{C\in\{candidates\}} \sum_{ngram\in C} Count_{clip}(ngram)}{\sum_{C\in\{candidates\}} \sum_{ngram\in C} Count(ngram)}, \tag{9}$$

$$Count_{clip} = min(Count, Max\_ref\_Count), \tag{10}$$

where *ngram* denotes the candidate ngrams, $Count_{clip}$ clips the total *Count* of each candidate ngram by the maximum number of overlap between *ngram* in the candidate and the references Max_ref_Count, to avoid overgenerating "reasonable" words. In brief, the numerator of p_n counts the number of candidate ngrams that appear in references, and the denominator counts all the candidate ngrams.

The next step is to compute a brevity penalty, which is to adapt the candidate translation to match the reference translation in length,

$$BP = \begin{cases} 1, & \text{if } l_c > l_r \\ e^{(1-l_r/l_c)}, & \text{if } l_c \le l_r, \end{cases} \tag{11}$$

where l_c is the length of a candidate translation and l_r is the length of the reference corpus. Then, we can get the BLEU score

$$\text{BLEU} = BP \cdot exp\left(\sum_{n=1}^{N} \frac{1}{N} logp_n\right). \tag{12}$$

In our experiment, we choose $N = 4$ to have the BLEU-4 score. Besides, we apply a smoothing method introduced by Lin et al. [38] to add one to the *ngram* hit count and total *ngram* count for $n > 1$. This way, candidate translations with less than *n* words can still get a positive score. We refer to the smoothed method as BLEUS-4 and use its implementation of NLTK[19] in our experiment.

4.2.2. ROUGE

The Recall-Oriented Understudy for Gisting Evaluation (ROUGE) was introduced by Lin et al. [39] to measure the quality of machine-generated summaries. It consists of several measures including ROUGE-N and ROUGE-L, which will be used in our experiment. On complementary of BLEU's bias on *ngram* precision, ROUGE-N focuses on the *ngram* recall, which is calculated as

$$\text{ROUGE-N} = \frac{\sum_{S\in\{References\}} \sum_{ngram\in S} Count_m(ngram)}{\sum_{S\in\{References\}} \sum_{ngram\in S} Count(ngram)}, \tag{13}$$

where *ngram* denotes the reference ngrams, $Count_m$ is the maximum number of overlap between *ngram* in a candidate summary and the references. In brief, the numerator of ROUGE-N is to count the number of overlap ngrams between candidates and references, and the denominator is to count all the reference ngrams.

ROUGE-L takes advantage of both the Longest Common Subsequence (LCS) and the F-measure to estimate the similarity between two summaries: the candidate summary S_{can} of length l_a and the reference summary S_{ref} of length l_e. The calculation is as follows:

$$Recall_{lcs} = \frac{LCS(S_{ref}, S_{can})}{l_e}, \tag{14}$$

Table 4
The evaluation results of models trained on the joint dataset of four programming languages. All the score numbers are averages over the tested posts of different languages.

Model	Language	BLEUS-4	ROUGE-1	ROUGE-2	ROUGE-L
Oracle	Java	54.58	83.76	65.13	83.31
	Python	51.43	82.40	62.55	81.81
	JS	53.10	83.02	63.68	82.55
	PHP	54.22	83.65	65.19	83.21
TF–IDF	Java	9.79	19.91	4.44	19.17
	Python	10.26	21.88	5.28	21.01
	JS	10.10	20.51	4.93	19.76
	PHP	10.49	21.24	5.15	20.30
$\text{BiLSTM}_{\text{joint}}$	Java	17.04	36.74	15.35	36.17
	Python	17.71	39.89	16.86	39.04
	JS	18.06	38.79	16.56	38.09
	PHP	18.66	39.97	17.99	38.92
$\text{BiLSTM-CC}_{\text{joint}}$	Java	19.73	41.10	19.54	40.04
	Python	19.74	42.67	19.97	41.72
	JS	20.59	42.62	20.36	41.59
	PHP	20.56	43.01	20.92	41.73
$\text{BART}_{\text{joint}}$	Java	20.80	44.21	21.12	42.42
	Python	21.01	45.69	22.44	44.28
	JS	21.54	45.65	22.29	43.81
	PHP	22.28	46.94	23.47	45.05
$\text{CCBERT}_{\text{joint}}$	Java	**21.16**	**44.26**	**21.58**	**42.92**
	Python	**22.40**	**46.88**	**22.89**	**44.92**
	JS	**22.18**	**45.72**	**22.40**	**44.15**
	PHP	**22.65**	**47.03**	**23.50**	**45.15**

$$Precision_{lcs} = \frac{LCS(S_{ref}, S_{can})}{l_a}, \tag{15}$$

$$\text{ROUGE-L} = \frac{Recall_{lcs}Precision_{lcs}}{Recall_{lcs} + Precision_{lcs}}, \tag{16}$$

In the end, we choose ROUGE-1, ROUGE-2, and ROUGE-L implemented by an open source library[20] for the evaluation metrics.

4.3. Model settings

We implement our encoder with the pre-trained parameters[21] of CodeBERT-base and keep its initial settings, where the vocabulary size is 50 265, the hidden size is 768, the dropout probability is 0.1, and the Transformer layer number is 12. Accordingly, we build a 12-layer decoder with randomly initialized parameters. Optimization is performed using the adaptive moment estimation (Adam) algorithm with $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-8}$ and $lr = 5 \times 10^{-5}$. We also apply a linear warm-up strategy to gradually increase the learning rate in the first 10% training steps. Four NVIDIA GeForce RTX 2080 Ti GPUs are used to train our model, where the training epoch is ten and batch size is 32. During decoding, we set the beam size to ten. We adjust all the hyperparameters to the validation set and report the evaluation results on the test set.

5. Results and analysis

In this section, we demonstrate the effectiveness of our model by conducting experiments to answer the following Research Questions (RQs):

RQ-1 Does our CCBERT model outperform the baseline models?

RQ-2 What is the advantage of using the bi-modal information of the entire question body?

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

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

[21] https://huggingface.co/microsoft/codebert-base.

Table 5
The evaluation results of models trained on the separate datasets of four programming languages. All the score numbers are averages over the tested posts of different languages. The Oracle and TF–IDF models are not affected by separated training.

Model	Language	BLEUS-4	ROUGE-1	ROUGE-2	ROUGE-L
BiLSTM_{sep}	Java	14.59	32.13	11.95	31.86
	Python	15.93	37.15	14.24	36.55
	JS	14.91	33.03	11.63	32.50
	PHP	13.03	28.41	08.73	27.62
BiLSTM-CC_{sep}	Java	18.22	38.89	18.09	38.21
	Python	18.84	41.49	18.98	40.56
	JS	19.35	40.92	18.28	40.15
	PHP	19.08	40.83	18.75	39.72
BART_{sep}	Java	19.32	42.52	19.93	41.66
	Python	20.43	44.93	21.77	43.97
	JS	20.19	43.54	20.56	42.70
	PHP	19.61	43.50	20.79	42.65
CCBERT_{sep}	Java	**20.90**	**43.06**	**21.15**	**41.76**
	Python	**22.02**	**46.69**	**22.55**	**44.86**
	JS	**21.24**	**44.55**	**21.05**	**42.80**
	PHP	**21.93**	**45.60**	**22.35**	**43.87**

RQ-3 How effective is our CCBERT model under low-resource circumstances?

RQ-4 How much influence does interrogative constraint have on model training?

RQ-5 How effective is our CCBERT model under human evaluation?

5.1. RQ-1: Does our CCBERT model outperform the baseline models?

Method: In order to investigate the superiority of our model, we compare it to the baselines mentioned in Section 4.1. We also apply two training strategies to the deep learning models. One is to train on the $Data_{exp}$ jointly with all the questions. The other is to train on separated smaller subsets of questions concerning different programming languages. Both training strategies share the same validation and test sets to compare the performance.

Tables 4 and 5 show the performance of all models on four evaluation metrics. In addition, we present four test examples in Table 6 to make intuitive comparisons.

Results: From Tables 4, 5, and 6, we have the following findings:

(1) The performance rankings are the same on both training strategies, where CCBERT outperforms all the baselines ranging from the retrieval-based model (TF–IDF) to the large-scale pre-trained model (BART). Moreover, all the models trained on the joint dataset perform better than those trained on separated subsets, attributing to the increased amount of training samples and the similar writing pattern shared by high-quality questions involving different programming languages. We have also noticed that Java questions are more difficult for all the models, which is similar to the results reported by Gao et al. [9]. This is partly because Java questions have a larger vocabulary than others, and models are more likely to encounter rare tokens.

(2) TF–IDF has the worst performance among all the baseline models and can barely compare with other baselines. This is not surprising because questions containing duplicated content in Stack Overflow have a high possibility of being closed, let alone only a small number of questions available in our train set. Besides, the nature of TF–IDF is a bag-of-word model, which does not take into account the overall meaning of the context, so it is barely possible for TF–IDF to retrieve the appropriate questions. All the samples in Table 6 show that the retrieved questions are totally different from the original ones.

(3) BiLSTM-CC and BiLSTM outperform TF–IDF by a large margin, indicating the superiority of neural generative models. Besides, BiLSTM-CC outperforms the vanilla BiLSTM by 11% on average on the joint dataset and by 29% on average on separated subsets, which proves the effectiveness of the copy and coverage mechanisms. From the samples in Table 6, we can find that BiLSTM often borrows the exact phrases from question bodies, while BiLSTM-CC can reorganize words into sentences. Despite the good performance of BiLSTM-CC, our CCBERT model outperforms it by 9% on average on the joint dataset and by 11% on average on separated subsets, indicating the superiority of Transformer-based models and the pre-training strategy. From the generated samples in Table 6, we can find that CCBERT is better at handling long-range dependencies than BiLSTM-CC. For instance, in the first sample, CCBERT notices that “this shape” refers to the “TriangleMesh” that appeared later in the question body, while BiLSTM-CC tends to focus on the content at the beginning of the question body. Furthermore, it is the same for the rest of the samples, where unwanted words (i.e., “android studio”, “spring boot”, and “crypto-stock”) in the front of the question body attract more attention from BiLSTM-CC. At the same time, CCBERT can find the critical words (i.e., “sqlite”, “service”, and “key–value”) that hide in the middle of the question body.

(4) BART is a competitive model, where CCBERT outperforms it by 1.3% on average on the joint dataset and by 3.6% on average on separated subsets. According to the samples in Table 6, BART is good at generating clear and readable titles because it is a generation-oriented model that has been pre-trained on a vast natural language corpus. However, we can see from the first and second samples that titles generated by BART miss the keywords (i.e., “TriangleMesh” and “sqlite”), we attribute this problem to BART’s inferior understanding of source code. For instance, in the first sample, BART cannot find that the “shape” at the beginning refers to the “TiangleMesh” object declared in the following code snippet. In the second sample, a major part of the body describes inserting data into the SQLite database, while BART only focuses on the unimportant word “android studio”. On the contrary, with the help of bi-modal pre-trained CodeBERT encoder, our CCBERT model better understands the source code and generates more semantic titles relevant to the original ones.

(5) The Oracle model has a surprisingly good performance on both subsets, which shows much space for improving current models. In terms of the recall-oriented ROUGE metric, the excellent performance of the Oracle model indicates that most tokens in a question title come from the corresponding question body. However, our CCBERT model can only identify a part of the useful tokens in question bodies, leading to a moderate performance on the BLEU metric. Nevertheless, we can find that all the titles generated by the Oracle model hardly adapt to the grammatical norms from the generated samples in Table 6, which indicates the necessity of applying generative models on this task. As for the reasons of the huge performance difference between our CCBERT model and the Oracle model, we think that our model may not have well handled the complex long-range bi-modal contexts, and the personalized writing habits of question titles also makes it hard for an automatic model to summarize in the same way as developers do.

Answer to RQ-1: Our CCBERT model outperforms the TF–IDF, BiLSTM, BiLSTM-CC, and BART models regarding all the automated evaluation metrics on both training strategies.

5.2. RQ-2: What is the advantage of using the bi-modal information of the entire question body?

Motivation: Although we have illustrated the necessity of using both text descriptions and code snippets to generate high-quality question titles, we would like to quantify the improvement of using the bi-modal information over the code-only setting in Gao et al.’s work [9].

Method: We post-process all question bodies in $Data_{exp}$ to keep code snippets and weed out text descriptions. We follow the jointly training strategy in this experiment. For the convenience of comparison, the new code-only dataset has questions in the same order as the previous joint

Table 6
The examples of our testing questions and automatically generated titles. Specifically, the green color marks the tokens appearing in original titles, the orange-red color marks the wrong focus, and the gray color marks the code snippets. The models with a *code* subscript in their names are trained on the code-only dataset.

Question body	Titles
I need to create this shape. I understand how to create simple shapes such as a cube, but I don't understand at all how to create such a shape. How to get the right points for these arrays? Please, help TriangleMesh mesh = new TriangleMesh(); mesh.getPoints().addAll(... mesh.getTexCoords().addAll(... //which points should be here mesh.getFaces().addAll(... //which points should be here return mesh;	**Origin:** how to create such shape using javafx trianglemesh **Oracle:** how to create such shape trianglemesh **TF–IDF:** how update the value in json file using java jackson **BiLSTM:** how to get the right points for arrays **BiLSTM-CC:** how to create such a shape in java **BART:** how to create such a shape **CCBERT:** how to create this shape using trianglemesh **BiLSTM-CC$_{code}$:** how to get points of a mesh **CCBERT$_{code}$:** how to add points to a trianglemesh

I am a beginner in mobile application building. I tried to put insert data function in my android studio but those insert function doesn't work and the input data can't be inserted...
I put code in MainActivity.java and DatabaseHelper.java. It doesn't give error report but when run the emulator and input data, my input can be inserted to sqlite database.
//oncreateMainActivity
super.onCreate(savedInstanceState...
myDb = new DatabaseHelper(...
submit2.setOnClickListener(...
//DatabaseHelper.java
public boolean insertData(String...
SQLiteDatabase db = this.getWritableDatabase(...
long result = db.insert(TABLE_NAME...
	Origin: how to insert data to sqlite through user input
	Oracle: to insert data to sqlite input
	TF–IDF: listview not show items stored in sqlite database
	BiLSTM: how to put function in my android studio
	BiLSTM-CC: how to insert data function in android studio
	BART: how to insert data in android studio
	CCBERT: how to insert input data to sqlite database
	
	BiLSTM-CC$_{code}$: how to add data to an activity in android
	CCBERT$_{code}$: how to get data from database in android

I have a simple web application where different users can log into it...send email of it's content to an outsider like third party....With all this, I am using Java Mail API to make it work and after hitting the send button,it sends directly to the recipient...Now, I want to modify this by doing this email feature as a service...the content and info filled in will be stored in a table in MYSQL and...
public void sendEmail(String ... {
Properties props = new Properties();
props.put("mail.smtp.host", host); //SMTP...
Authenticator auth = new Authenticator()...
Can this be done in the way...how to make it work?
	Origin: java - how to use services for sending email
	Oracle: java how to sending email
	TF–IDF: gae send email from gmail account
	BiLSTM: how can i send email to an outsider
	BiLSTM-CC: how to send email from database in java
	BART: how to send email using java mail api
	CCBERT: how to send email using services in java
	
	BiLSTM-CC$_{code}$: how to set the header of a mail in a mail
	CCBERT$_{code}$: how to send email using java mail

I want to work with crypto-stock data described here in my spring boot application. The RESTTemplate uses Gson for deserialization. Response data looks like: {"IOST": {"EUR": 0.01147, I have already...problem is that this comes as a single object with key–value pairs insted of as an array. The result should be a list of following objects: public class Symbol { private Long id; private String symbol... Any idea how this can be accomplished this?	**Origin:** how to deserialize a key--value map to a list **Oracle:** how to a key–value to a list **TF–IDF:** bigdecimal not keeping actual value when returned **BiLSTM:** how to work with crypto-stock data **BiLSTM-CC:** how to parse json data in spring boot **BART:** how to deserialize crypto-stock data **CCBERT:** how to deserialize key–value pairs **BiLSTM-CC$_{code}$:** how to convert json object to java object **CCBERT$_{code}$:** how to deserialize a json object in java

Table 7
The performance of CCBERT and BiLSTM-CC on the code-only dataset. All the score numbers are averages over the tested posts of different languages.

Model	Language	BLEUS-4	ROUGE-1	ROUGE-2	ROUGE-L
BiSLTM-CC$_{code}$	Java	11.78	25.04	7.15	25.50
	Python	13.38	30.54	9.78	30.48
	JS	13.02	28.00	8.35	28.18
	PHP	13.13	29.25	9.09	28.63
CCBERT$_{code}$	Java	12.84	28.73	9.62	28.58
	Python	14.03	33.35	11.67	32.67
	JS	13.67	30.57	10.36	30.20
	PHP	14.32	32.80	11.90	31.88

dataset during training and testing. We choose BiLSTM-CC and CCBERT as representatives of our generative models and show their performance in Table 7.

Results: There is a severe decline in the performance of both models when using only code snippets for training. Specifically, the performance of CCBERT declines by 37% on average, and BiLSTM-CC drops its performance by 36% on average. Such results are expected because code snippets themselves cannot offer sufficient context to a question.

According to the samples in Table 6, it is hard to tell the corresponding titles of all the four samples given only code snippets. Therefore, the generated titles may be incomplete and incorrect. For instance, in the first sample, both BiLSTM-CC$_{code}$ and CCBERT$_{code}$ pay attention to the "TriangleMesh", but neither of them deduces the word "shape" used in the title. In the second sample, both models fail to tell that the actual purpose of using "Android activity" and "SQLite database" is to insert user input data into the SQLite database. In the third and fourth samples, although both models manage to create the words (i.e., "using", "convert", "deserialize") that are not in the code snippets, there are still few overlaps between the generated words and the wanted ones. However, under code-only circumstances and without changing model structure and hyperparameters, CCBERT$_{code}$ shows the superiority over BiLSTM-CC$_{code}$, which also indicates its generalization ability on different tasks.

Table 8
The performance of CCBERT and BiLSTM-CC on the datasets with different sizes. All the score numbers are averages over the tested posts of different languages.

Model	Language	BLEUS-4	ROUGE-1	ROUGE-2	ROUGE-L
BiLSTM-CC$_{\text{joint}}$	Java	19.73	41.10	19.54	40.04
	Python	19.74	42.67	19.97	41.72
	JS	20.59	42.62	20.36	41.59
	PHP	20.56	43.01	20.92	41.73
BiSLTM-CC$_{\text{joint/2}}$	Java	18.99	39.92	18.05	39.20
	Python	19.52	42.00	19.36	41.40
	JS	19.58	41.21	18.76	40.50
	PHP	20.06	41.83	20.03	41.05
BiSLTM-CC$_{\text{joint/4}}$	Java	18.74	39.73	17.81	39.09
	Python	19.28	41.93	18.82	41.11
	JS	19.43	40.83	18.41	40.06
	PHP	20.02	41.39	19.70	40.83
BiSLTM-CC$_{\text{joint/8}}$	Java	17.34	38.14	16.50	37.64
	Python	17.61	39.98	17.12	39.52
	JS	18.39	39.60	17.32	39.16
	PHP	18.72	40.34	18.20	39.56
CCBERT$_{\text{joint}}$	Java	21.16	44.26	21.58	42.92
	Python	22.40	46.88	22.89	44.92
	JS	22.18	45.72	22.40	44.15
	PHP	22.65	47.03	23.50	45.15
CCBERT$_{\text{joint/2}}$	Java	21.02	43.78	20.78	42.08
	Python	21.83	45.84	22.21	44.14
	JS	21.89	44.75	21.79	42.85
	PHP	22.64	46.21	23.24	44.27
CCBERT$_{\text{joint/4}}$	Java	20.56	43.45	20.52	41.83
	Python	21.22	45.81	21.72	44.02
	JS	21.55	44.68	21.45	42.76
	PHP	22.15	46.06	22.54	44.19
CCBERT$_{\text{joint/8}}$	Java	20.45	43.06	20.17	41.45
	Python	20.73	44.42	20.92	43.56
	JS	20.89	43.44	20.36	41.75
	PHP	21.95	45.12	22.26	43.56

Table 9
The performance of CCBERT and BiLSTM-CC trained on $Data_{exp+}$ without the interrogative constraint. All the score numbers are averages over the tested posts of different languages.

Model	Language	BLEUS-4	ROUGE-1	ROUGE-2	ROUGE-L
BiSLTM-CC$_{\text{exp+}}$	Java	16.71	31.79	14.52	30.36
	Python	17.40	33.61	14.70	31.65
	JS	17.78	33.67	15.38	32.16
	PHP	18.27	34.32	15.46	32.19
CCBERT$_{\text{exp+}}$	Java	18.68	36.32	16.43	33.81
	Python	19.18	37.50	16.74	34.77
	JS	19.55	37.81	17.04	35.31
	PHP	20.19	39.02	17.05	35.69

> Answer to RQ-2: Applying bi-modal information greatly boosts both models' performance, where CCBERT still outperforms BiLSTM-CC.

5.3. RQ-3: How effective is our CCBERT model under low-resource circumstances?

Motivation: Data-hungry is a common issue in the field of deep learning, which significantly hinders the application of many excellent models. Our model may need to train on massive high-quality questions. Since previous experiments have proved that CCBERT can handle the situation with around 200,000 samples for training, we carry out this experiment to discuss the effectiveness of our model under low-resource circumstances.

Method: We first make several copies of $Data_{exp}$, and then randomly erase a certain amount of questions in the train sets, leaving the validation and test sets untouched. We choose three fractions as the percentage of samples to erase, which are 1/2, 3/4, and 7/8. This makes three new train sets sized of 98,909 (Data$_{\text{exp}}$/2), 49,454 (Data$_{\text{exp}}$/4), and 24,727 (Data$_{\text{exp}}$/8). Along with the CCBERT model, we also train BiLSTM-CC on these datasets for comparison. Table 8 presents the experimental results on the datasets with different sizes.

Table 10
The partition size of $Data_{exp+}$.

Language	Train	Validation	Test
Java	183,443	7000	7000
Python	207,323	7000	7000
JS	174,374	7000	7000
PHP	104,227	4000	4000
Total	669,367	25,000	25,000

Results: It is as expected that both models have suffered performance degradation on smaller datasets. To our surprise, even if the amount of data decreases exponentially, the performance has a steady decline. Specifically, the performance of CCBERT declines by 2% on average on the Data$_{\text{exp}}$/2 subset, by 2.8% on the Data$_{\text{exp}}$/4 subset, and by 4.8% on the Data$_{\text{exp}}$/8 subset; while BiLSTM-CC drops its performance by 3% on average on the Data$_{\text{exp}}$/2 subset, by 3.8% on the Data$_{\text{exp}}$/4 subset, and by 8.2% on the Data$_{\text{exp}}$/8 subset. It indicates that our task is not so sensitive to the data volume and further verifies the existence of a writing pattern shared by high-quality questions. Meanwhile, developers have personalized writing habits, so a more considerable amount of data can help eliminate such noise and improve the performance of our model. Facilitated by the pre-trained CodeBERT encoder, our CCBERT model is better initialized to resist data noise and requires fewer data. We can see from the results that with only one-eighth of the data, CCBERT still outperforms BiLSTM-CC trained on the full dataset.

> Answer to RQ-3: Compared to BiLSTM-CC, our CCBERT model shows significant superiority under low-resource circumstances.

5.4. RQ-4: How much influence does interrogative constraint have on model training?

Motivation: Applying the interrogative constraint may reduce the data noise and make it easier to train our models. Nevertheless, our dataset could be narrowed and biased because of this. So we carry out this experiment to investigate the actual influence of interrogative constraint on our model's performance.

Method: We first build a dataset $Data_{exp+}$ in the same way as building $Data_{exp}$, except for applying the interrogative constraint. This way, we get a much larger dataset. Then, we train and evaluate both BiLSTM-CC and CCBERT models on the new dataset, following the jointly training strategy. The automated evaluation results are shown in Table 9. The statistics of $Data_{exp+}$ is shown in Table 10.

Results: We believe that a question's popularity does not necessarily attribute to a unified title format. Many other aspects like the clarity of question description and the popularity of the asked domain will influence the popularity of a question. We have manually studied the questions that we regard as high-quality ones and have no interrogatives in their titles. We find that those titles are always casually written. For example, a title can be a combination of keywords,[22] a short phrase,[23] an error message,[24] etc.

[22] https://stackoverflow.com/questions/53279561/java-month-enum.

[23] https://stackoverflow.com/questions/53218222/capture-logs-in-a-test.

[24] https://stackoverflow.com/questions/53344676/java-lang-illegalstateexception-inputstream-has-already-been-read-do-not-use.

Table 11
Human evaluation results of the TF–IDF, BiLSTM-CC, and CCBERT models trained on the joint dataset of four languages. The ratios of four different scores and the average score are listed grouped by different criteria.

Criteria	Model	Score 1	Score 2	Score 3	Score 4	Avg Score
Readability	TF–IDF	–	16.6%	81.2%	2.2%	2.856
	BiLSTM-CC$_{\text{joint}}$	–	23.6%	76.2%	0.2%	2.766
	CCBERT$_{\text{joint}}$	–	19.2%	80.2%	0.6%	2.814
Correlation	TF–IDF	91.6%	8.4%	–	–	1.084
	BiLSTM-CC$_{\text{joint}}$	27.6%	55.4%	14.2%	2.8%	1.922
	CCBERT$_{\text{joint}}$	16.8%	47.4%	30.8%	5%	2.240

According to Table 9, both models have poor performance on $Data_{exp+}$, despite having so much data for training. Specifically, the CCBERT model has an average 19.4% lower evaluation score than using $Data_{exp}$, while it is 21.4% for BiLSTM-CC. But in this experiment, our CCBERT model still outperforms BiLSTM-CC by 11.6% on average, which verifies the superiority of our model under different circumstances.

> Answer to RQ-4: Dropping the interrogative constraint will lead to a decline in the performance of both CCBERT and BiLSTM-CC models.

5.5. RQ-5: How effective is our CCBERT model under human evaluation?

Motivation: Automated evaluation is not always trustworthy because it is hard to decide the actual human-perceived quality in different situations. The BLEU and ROUGE metrics used in this study mainly focus on the lexical ngram overlap between text sequences ignoring the grammatical correctness and the semantic similarity. So in this experiment, we will perform a more human-centered evaluation to investigate the overall quality of the titles generated by our models.

Method: We consider the two criteria introduced in Table 1 when manually evaluating the generated titles, either of them can be scored between 1 and 4. We randomly sample 500 questions in the test set of $Data_{exp}$ and then obtain 1500 titles generated by the TF–IDF, BiLSTM-CC$_{\text{joint}}$, and CCBERT$_{\text{joint}}$ models. We invite five graduate students who are not co-authors to help us with this experiment. They are all experienced programmers and familiar with Stack Overflow. Each participant is assigned 100 questions, and we attach each question with three generated titles. The participants need to rate the titles based on the scoring standards manually, and they are blinded as to which title is generated by our model. The evaluation results are shown in Table 11.

Results: In terms of the readability criteria, the performance of the three models is evenly matched. As expected, TF–IDF achieves the highest score because it retrieves and returns the titles written by developers. Our CCBERT model outperforms BiLSTM-CC by only 1.7% and is only 1.5% less good than TF–IDF. To conclude, most of the generated titles by our CCBERT model are regarded as easy to read and understand.

Regarding the correlation criteria, both the CCBERT and BiLSTM-CC models outperform TF–IDF by a large margin. It shows the superiority of generative models over the retrieval-based method on semantic understanding. The performance of our CCBERT model is 16.5% better than BiLSTM-CC, which is not surprising because attributed to the pre-trained CodeBERT encoder, our model is more capable of handling long-range dependencies in bi-modal content. However, according to Table 11, only less than a half of the generated titles of CCBERT are considered well matching the questions. It suggests that there is still much room for improvements in our approach.

> Answer to RQ-5: CCBERT performs much better than TF–IDF and BiLSTM-CC concerning the correlation criteria but a little bit worse than human written titles retrieved by TF–IDF concerning the readability criteria under human evaluation.

6. Related work

Since we treat our TGEQB task as abstractive summarization, we take the related works in text summarization and code summarization for reference. We briefly introduce the recent literature in this section.

6.1. Text summarization

There are extractive and abstractive ways for summarization tasks. Both ways have been attracting extensive research interest.

The extractive models select sentences or paragraphs from source texts to best match the target summary. The idea of using a hierarchical encoder and an extractor for document summarization was proposed by Cheng et al. [40]. Later, researchers have proposed various solutions to deal with different detailed problems. For example, Zhou et al. [41] argued that one should not separate the sentence scoring and selection steps, so they proposed an integrated model to merge the two steps. Xu et al. [42] argued that BERT-based models could not capture dependencies among discourse units, which leads to the problem of having unwanted phrases in extracted summaries. To tackle this problem, they proposed to encode the rhetorical structure theory trees with a graph convolutional network. Jia et al. [43] argued that BERT-based models neglect the inherent dependencies among reference sentences, and they proposed to refine the sentence representations with a redundancy aware graph attention network. These novel models performed well on semantic parsing. However, our task requires the model to give readable titles, where the extractive ways have been proved unworkable on our dataset (reference the Oracle method's performance in Table 6).

In general, abstractive models are not restricted to selecting and rearranging the original text but to generating each word from a given vocabulary. See et al. [26] argued that vanilla attentional sequence-to-sequence models always produce inaccurate factual details and duplicate phrases. So they proposed to use a hybrid generator incorporating both the copy and coverage mechanisms. Gehrmann et al. [24] found the problem that abstractive models were poor at content selection. Instead of adding fancy mechanisms, they proposed a two-stage process to train an extractor and then use it as bottom-up attention to guide the generator. Liu et al. [25] extended this idea by using a two-stage fine-tuning on both extraction and generation tasks. In addition, they proposed to use different optimizers for the encoder and decoder to alleviate mismatch brought by different objectives. Lewis et al. [20] proposed BART. This generative-oriented pre-training model has achieved excellent performance in abstractive summarization tasks, so we choose it as a baseline model to compare with ours. Abstractive summarization is similar to our task in many aspects. However, our dataset is more challenging due to the complex bi-modal context and the difficult rare tokens, which is why we adopt the CodeBERT model and the copy mechanism.

6.2. Code summarization

Code summarization aims to generate readable and meaningful comments that accurately describe the given programs or subroutines, which is very useful for code search and comprehension.

One way to deal with source code is to treat it as a sequence. Iyer et al. [44] first proposed to use attentional LSTMs to produce summaries describing code snippets, and they released their training corpus. Hu et al. [45] further looked into the possibility of using API knowledge to generate comments that better describe the functionality of source code. Wei et al. [46] proposed to use comments of existing similar source code to guide new comment generation. Wei et al. [47] exploited the relations between code summarization and code generation, and proposed a dual framework to train the two tasks simultaneously. The experimental results showed that performance improvements were achieved on both tasks. Hu et al. [48] argued that code tokens should not be processed sequentially. Hence, they proposed an abstract syntax tree-based structural code representation and verified its effectiveness in generating code comments. Ahmad et al. [49] first introduced the Transformer model to this task. They proposed to use a pairwise position encoding to capture the long-range dependencies among code tokens. The above approaches treated source code as text sequences, but they also valued the particular information hidden behind the code. Their experiments convinced us that the programming language is different from the natural language.

The other way is to convert the source code into other forms of representation. Wan et al. [50] used a sequential encoder as well as a tree-based encoder to capture the general information from code. They also applied an actor–critic network to overcome the exposure bias issue of the auto-regressive decoder. LeClair et al. [51] also used dual encoders and incorporated the copy mechanism to reserve necessary tokens reported by the AST analyzer. Besides, they [52] further proposed to use a graph-based neural architecture that achieved even better performance. Yang et al. [53] employed a sequential encoder and a graph-based encoder to learn the global and local semantic information to generate code comments of smart contracts. The studies mentioned above show that source code needs technological transformation for models to extract the semantic information for summarization. Unfortunately, we find that content marked with the "<code>" tag in our filtered SO questions are not always syntactically correct source code. Therefore, we treat the code snippets as a sequence of tokens and use CodeBERT as the encoder, which is code-aware and takes sequences as input.

7. Threats to validity

In this section, we identify the potential threats that might affect the recurrence of our experiments and the validation of our results.

The threats to internal validity concern us in two aspects, one is the re-implementation of baselines, the other is the design of the CCBERT model. To address the first issue, we rebuild the default development environment and choose the recommended settings for baseline models. As for the second issue, we have made trade-offs between different techniques. For example, we give up using the coverage mechanism because it is incompatible with the parallel decoding fashion of our Transformer decoder. Further experiments also show that our model is not troubled by the repetition problem. Besides, training the large CCBERT model with insufficient data may cause the overfitting issue. To address this issue, our CodeBERT encoder has initialized its parameters through self-supervised learning with massive data in the pre-training stage. Furthermore, in our experiments, we train our model with a small learning rate, which also helps alleviate the problem of overfitting.

The threats to external validity primarily relate to the quality and generalizability of our dataset. We notice that the SOTorrent dataset proposed by Baltes et al. [54] shares a lot in common with ours. SOTorrent aims to provide access to the version history of SO content, which involves legacy formats issues and contains a lot of duplicate posts. We build our own dataset, because ours is directly extracted from the latest version of SO posts, which has a unified HTML-style format and can be easily parsed into text/code blocks by a naive HTML parser. There may be a deviation between our dataset and the realistic data. To make our dataset more realistic, we build it concerning four programming languages, apply two training strategies for comparison, and choose the lately posted questions for testing.

The threats to construct validity mainly relate to the evaluation measures. Though the BLEU and ROUGE metrics have been widely used, automated evaluation is still an open problem in the domain of text generation [55–57]. We perform a human-centered evaluation in terms of the readability and correlation criteria to address this issue.

8. Conclusion and future work

In this paper, we propose a new task to summarize question titles from bi-modal context and a novel model named CCBERT to tackle this problem. CCBERT incorporates the copy mechanism and the CodeBERT model, which can handle rare tokens and capture the long-range dependencies between bi-modal tokens. We build a large-scale dataset with sufficient high-quality questions concerning four programming languages. We choose the BLEU and ROUGE metrics for automated evaluation and various baseline models for comparison. Both automated and human evaluation results demonstrate the superiority of our model. We have released our dataset and source code for follow-up researches.

For future work, we will try to tackle the problem of handling very long sequences. In addition, we consider using Incremental Learning techniques to make our model continuously learn new knowledge from new samples and retain most of the knowledge already learned.

Acknowledgments

This work is supported in part by the Natural Science Foundation of Chongqing City, China (No. cstc2021jcyj-msxmX1115), the Fundamental Research Funds for the Central Universities (WUT: 213110003 and 223110002), the project supported by Sanya Science and Education Innovation Park of Wuhan University of Technology (No. 2020KF0059), the General Research Fund of the Research Grants Council of Hong Kong (No. 11208017), the research funds of City University of Hong Kong (No. 7005028 and 7005217), and the Research Support Fund by Intel, USA (No. 9220097), and funding supports from other industry partners (No. 9678149, 9440227, 9229029, 9440180, and 9220103). We thank Guoping Nie for his helpful comments about this work.

References

[1] P. Chakraborty, R. Shahriyar, A. Iqbal, G. Uddin, How do developers discuss and support new programming languages in technical Q&A site? An empirical study of go, swift, and rust in stack overflow, Inf. Softw. Technol. 137 (2021) 106603.

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

[3] G. Uddin, F. Khomh, C. Roy, Mining API usage scenarios from stack overflow, Inf. Softw. Technol. 122 (2020) 106277.

[4] S. Mondal, C.K. Saifullah, A. Bhattacharjee, M.M. Rahman, C.K. Roy, 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), 2021, pp. 1–11.

[5] J.E. Montandon, C. Politowski, L.L. Silva, M.T. Valente, F. Petrillo, Y.-G. Guéhéneuc, What skills do IT companies look for in new developers? A study with stack overflow jobs, Inf. Softw. Technol. 129 (2021) 106429.

[6] A. Tahir, J. Dietrich, S. Counsell, S. Licorish, A. Yamashita, A large scale study on how developers discuss code smells and anti-pattern in stack exchange sites, Inf. Softw. Technol. 125 (2020) 106333.

[7] S. Wang, T.-H. Chen, A.E. Hassan, How do users revise answers on technical Q&A websites? A case study on stack overflow, IEEE Trans. Softw. Eng. 46 (9) (2018) 1024–1038.

[8] H. Wang, B. Wang, C. Li, L. Xu, J. He, M. Yang, SOTagRec: A combined tag recommendation approach for stack overflow, in: Proceedings of the 2019 4th International Conference on Mathematics and Artificial Intelligence, 2019, pp. 146–152.

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

[10] P. Arora, D. Ganguly, G.J. Jones, The good, the bad and their kins: Identifying questions with negative scores in stackoverflow, in: 2015 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM), IEEE, 2015, pp. 1232–1239.

[11] F. Calefato, F. Lanubile, N. Novielli, How to ask for technical help? Evidence-based guidelines for writing questions on stack overflow, Inf. Softw. Technol. 94 (2018) 186–207.

[12] D. Correa, A. Sureka, Fit or unfit: analysis and prediction of'closed questions' on stack overflow, in: Proceedings of the First ACM Conference on Online Social Networks, 2013, pp. 201–212.

[13] Y. Yao, H. Tong, T. Xie, L. Akoglu, F. Xu, J. Lu, Want a good answer? Ask a good question first!, 2013, arXiv preprint arXiv:1311.6876.

[14] J. Gu, Z. Lu, H. Li, V. Li, Incorporating copying mechanism in sequence-to-sequence learning, 2016, ArXiv arXiv:1603.06393.

[15] Z. Tu, Z. Lu, Y. Liu, X. Liu, H. Li, Modeling coverage for neural machine translation, Comput. Lang. (2016) arXiv.

[16] U. Khandelwal, H. He, P. Qi, D. Jurafsky, 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), 2018, pp. 284–294.

[17] Z. Feng, D. Guo, D. Tang, N. Duan, X. Feng, M. Gong, L. Shou, B. Qin, T. Liu, D. Jiang, M. Zhou, CodeBERT: A pre-trained model for programming and natural languages, in: FINDINGS, 2020.

[18] H.P. Luhn, The automatic creation of literature abstracts, IBM J. Res. Dev. 2 (1958) 159–165.

[19] M. Schuster, K.K. Paliwal, Bidirectional recurrent neural networks, IEEE Trans. Signal Process. 45 (1997) 2673–2681.

[20] M. Lewis, Y. Liu, N. Goyal, M. Ghazvininejad, A. Mohamed, O. Levy, V. Stoyanov, L. Zettlemoyer, BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension, in: ACL, 2020.

[21] Z. Dai, Z. Yang, Y. Yang, J.G. Carbonell, Q.V. Le, R. Salakhutdinov, Transformer-XL: Attentive language models beyond a fixed-length context, 2019, ArXiv arXiv:1901.02860.

[22] I. Beltagy, M.E. Peters, A. Cohan, Longformer: The long-document transformer, 2020, ArXiv arXiv:2004.05150.

[23] M. Zaheer, G. Guruganesh, K.A. Dubey, J. Ainslie, C. Alberti, S. Ontañón, P. Pham, A. Ravula, Q. Wang, L. Yang, A. Ahmed, Big bird: Transformers for longer sequences, 2020, ArXiv arXiv:2007.14062.

[24] S. Gehrmann, Y. Deng, A.M. Rush, Bottom-up abstractive summarization, in: EMNLP, 2018.

[25] Y. Liu, M. Lapata, Text summarization with pretrained encoders, in: EMNLP/IJCNLP, 2019.

[26] A. See, P.J. Liu, C.D. Manning, Get to the point: Summarization with pointer-generator networks, in: ACL, 2017.

[27] C. Pan, M. Lu, B. Xu, An empirical study on software defect prediction using CodeBERT model, Appl. Sci. 11 (2021) 4793.

[28] K. Zhao, Z. Xu, M. Yan, T. Zhang, D. Yang, W. Li, A comprehensive investigation of the impact of feature selection techniques on crashing fault residence prediction models, Inf. Softw. Technol. 139 (2021) 106652.

[29] K. Zhao, J. Liu, Z. Xu, L. Li, M. Yan, J. Yu, Y. Zhou, Predicting crash fault residence via simplified deep forest based on a reduced feature set, in: 2021 IEEE/ACM 29th International Conference on Program Comprehension (ICPC), 2021, pp. 242–252.

[30] K. Zhao, Z. Xu, T. Zhang, Y. Tang, Simplified deep forest model based just-in-time defect prediction for android mobile apps, in: 2020 IEEE 20th International Conference on Software Quality, Reliability and Security (QRS), 2020, p. 222.

[31] K. Zhao, J. Liu, Z. Xu, X. Liu, L. Xue, Z. Xie, Y. Zhou, X. Wang, Graph4Web: A relation-aware graph attention network for web service classification, J. Syst. Soft. (ISSN: 0164-1212) (2022) 111324, http://dx.doi.org/10.1016/j.jss.2022.111324.

[32] E. Mashhadi, H. Hemmati, Applying CodeBERT for automated program repair of java simple bugs, in: 2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR), 2021, pp. 505–509.

[33] A. Vaswani, N.M. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A.N. Gomez, L. Kaiser, I. Polosukhin, Attention is all you need, 2017, ArXiv arXiv:1706.03762.

[34] D. Bahdanau, K. Cho, Y. Bengio, Neural machine translation by jointly learning to align and translate, 2015, CoRR arXiv:1409.0473.

[35] J. Devlin, M.-W. Chang, K. Lee, K. Toutanova, BERT: Pre-training of deep bidirectional transformers for language understanding, in: NAACL, 2019.

[36] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever, Language models are unsupervised multitask learners, 2019.

[37] K. Papineni, S. Roukos, T. Ward, W.-J. Zhu, Bleu: a method for automatic evaluation of machine translation, in: ACL, 2002.

[38] C.-Y. Lin, F. Och, ORANGE: a method for evaluating automatic evaluation metrics for machine translation, in: COLING, 2004.

[39] C.-Y. Lin, ROUGE: A package for automatic evaluation of summaries, in: ACL 2004, 2004.

[40] J. Cheng, M. Lapata, Neural summarization by extracting sentences and words, 2016, ArXiv arXiv:1603.07252.

[41] Q. Zhou, N. Yang, F. Wei, S. Huang, M. Zhou, T. Zhao, Neural document summarization by jointly learning to score and select sentences, in: ACL, 2018.

[42] J. Xu, Z. Gan, Y. Cheng, J. Liu, Discourse-aware neural extractive text summarization, in: ACL, 2020.

[43] R. Jia, Y. Cao, H. Tang, F. Fang, C. Cao, S. Wang, Neural extractive summarization with hierarchical attentive heterogeneous graph network, in: EMNLP, 2020.

[44] S. Iyer, I. Konstas, A. Cheung, L. Zettlemoyer, Summarizing source code using a neural attention model, in: ACL, 2016.

[45] X. Hu, G. Li, X. Xia, D. Lo, S. Lu, Z. Jin, Summarizing source code with transferred API knowledge, in: IJCAI, 2018.

[46] B. Wei, Retrieve and refine: Exemplar-based neural comment generation, in: 2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE), 2019, pp. 1250–1252.

[47] B. Wei, G. Li, X. Xia, Z. Fu, Z. Jin, Code generation as a dual task of code summarization, 2019, ArXiv arXiv:1910.05923.

[48] X. Hu, G. Li, X. Xia, D. Lo, Z. Jin, Deep code comment generation, in: 2018 IEEE/ACM 26th International Conference on Program Comprehension (ICPC), 2018, pp. 200–20010.

[49] W.U. Ahmad, S. Chakraborty, B. Ray, K.-W. Chang, A transformer-based approach for source code summarization, in: ACL, 2020.

[50] Y. Wan, Z. Zhao, M. Yang, G. Xu, H. Ying, J. Wu, P.S. Yu, Improving automatic source code summarization via deep reinforcement learning, in: 2018 33rd IEEE/ACM International Conference on Automated Software Engineering (ASE), 2018, pp. 397–407.

[51] A. LeClair, S. Jiang, C. McMillan, A neural model for generating natural language summaries of program subroutines, in: 2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE), 2019, pp. 795–806.

[52] A. LeClair, S. Haque, L. Wu, C. McMillan, Improved code summarization via a graph neural network, in: Proceedings of the 28th International Conference on Program Comprehension, 2020.

[53] Z. Yang, J. Keung, X. Yu, X. Gu, Z. Wei, X. Ma, M. Zhang, A multi-modal transformer-based code summarization approach for smart contracts, in: 29th IEEE/ACM International Conference on Program Comprehension, ICPC 2021, Madrid, Spain, May 20-21, 2021, IEEE, 2021, pp. 1–12.

[54] S. Baltes, L. Dumani, C. Treude, S. Diehl, SOTorrent: Reconstructing and analyzing the evolution of stack overflow posts, in: 2018 IEEE/ACM 15th International Conference on Mining Software Repositories (MSR), 2018, pp. 319–330.

[55] T. Sellam, D. Das, A.P. Parikh, BLEURT: Learning robust metrics for text generation, in: ACL, 2020.

[56] Y.-T. Yeh, M. Eskénazi, S. Mehri, A comprehensive assessment of dialog evaluation metrics, 2021, ArXiv arXiv:2106.03706.

[57] A.R. Fabbri, W. Kryscinski, B. McCann, R. Socher, D. Radev, SummEval: Re-evaluating summarization evaluation, Trans. Assoc. Comput. Linguist. 9 (2021) 391–409.