SOTitle: A Transformer-based Post Title Generation Approach for Stack Overflow

Ke Liu[†], Guang Yang[†], Xiang Chen[†‡*], Chi Yu[†]
[†]*School of Information Science and Technology*, *Nantong University*, China
[‡]*State Key Laboratory of Information Security*, *Institute of Information Engineering, Chinese Academy of Sciences*, China
Email: aurora.ke.liu@outlook.com, novelyg@outlook.com, xchencs@ntu.edu.cn, yc_struggle@163.com

***Abstract*—On Stack Overflow, developers can not only browse question posts to solve their programming problems but also gain expertise from the question posts to help improve their programming skills. Therefore, improving the quality of question posts in Stack Overflow has attracted the wide attention of researchers. A concise and precise title can play an important role in helping developers understand the key information of the question post, which can improve the post quality. However, the quality of the generated title is not high due to the lack of professional knowledge related to their questions or the poor presentation ability of developers. A previous study aimed to automatically generate the title by analyzing the code snippets in the question post. However, this study ignored the useful information in the corresponding problem description. Therefore, we propose an approach SOTitle for automatic post title generation by leveraging the code snippets and the problem description in the question post (i.e., the multi-modal input). SOTitle follows the Transformer structure, which can effectively capture long-term dependencies through a multi-head attention mechanism. To verify the effectiveness of SOTitle, we construct a large-scale high-quality corpus from Stack Overflow, which includes 1,168,257 high-quality question posts for four popular programming languages. Experimental results show that SOTitle can significantly outperform six state-of-the-art baselines in both automatic evaluation and human evaluation. To encourage follow-up studies, we make our corpus and approach publicly available.**

***Index Terms*—Post title generation, Question post quality, Stack Overflow mining, Transformer, Code snippet, Problem description**

I. Introduction

On Stack Overflow, developers can post their problems and wait for other community members to give corresponding answers to their problems. When developers encounter similar problems, such problems and corresponding answers are valuable and can be reused. Until now, millions of developers use Stack Overflow to search for high-quality answers for their programming problems. Moreover, Stack Overflow becomes a knowledge base for developers to learn programming skills by browsing high-quality posts [1] [2] [3] [4].

While the number of question posts in Stack Overflow has been growing rapidly, there are still a large number of problems, which have not received high-quality answers. These problems may be unclear, unspecific, difficult to understand, or unattractive to related developers [5].

These low-quality question posts not only fail to get effective help, but also hinder the process of knowledge generation [6] [7]. To solve this problem, previous studies have been conducted on the quality prediction of the question posts [1], [8]–[13]. For example, Correa and Sureka [10] surveyed the closed problems in Stack Overflow and found that high-quality problems should contain enough code for others to reproduce the problem.

Except for the prediction of low-quality question posts and how to improve the performance of the quality prediction models, some studies [10], [11], [14] found that an important reason for low-quality question posts is that developers do not create informative post titles. Some developers may lack knowledge related to their problems, or they may have poor presentation ability. For these developers, writing high-quality problem titles is a challenging task. Therefore, automatically generating post titles for Stack Overflow is needed.

To our best knowledge, Gao et al. [1] were the first to automatically generate a post title according to a specific code snippet. However, they ignored the valuable information in the corresponding problem description. As shown in Fig. 1, there are two posts in Stack Overflow, which have the same code snippet. However, these two posts have different problem descriptions, which result in different post titles. According to this example, we can find the problem description of the post can provide valuable information, which sometimes cannot be provided by the code snippet for title generation. Therefore, considering both modalities (i.e., code snippet and problem description) as the input can help generate high-quality post titles.

The main ideas of our proposed approach SOTitle have three aspects. (1) We model the multi-modal input by concatenating the code snippet and the problem description together and follows the Transformer structure. (2) We formalize question post title generation for each programming language as separate but related tasks. Since related tasks can improve each other's performance by using shared and complementary information, we utilize multi-task learning [15], which can guarantee the generalization of our approach by learning multiple tasks simultaneously. (3) We use the method SentencePiece [16] to split the code snippet and the problem description to alleviate the OOV (out-of-vocabulary) issue in the post title generation problem. SentencePiece is designed for neural machine translation and it is a language-independent

* Xiang Chen is the corresponding author.

978-1-6654-3786-8/22/$31.00 ©2022 IEEE
DOI 10.1109/SANER2022.2022.00075

Code Snippet(Python):

```
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
```

Problem Description (Python) 1:
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.
Title 1:
Where in my django app do I implement this get_client_ip() function?
URL 1:
https://stackoverflow.com/questions/57107116/

Problem Description (Python) 1:
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.

Title 2:
How get the "real" user ip address?

URL 2:
https://stackoverflow.com/questions/13076259/

Fig. 1. Two posts from Stack Overflow, which have the same code snippet but different problem descriptions

subword tokenizer and detokenizer.

Since there is no readily available corpus, to verify the effectiveness of our proposed approach, we gathered 1,168,257 high-quality problem posts for four different programming languages (i.e., Java, C#, Python, and JavaScript) from Stack Overflow. Furthermore, we compare `SOTitle` with six state-of-the-art baselines via automatic evaluation (i.e., Rouge [17]) and human study. Specifically, we first select the state-of-the-art post title generation approach `Code2Que` proposed by Gao et al. [1] as the first baseline. Then we select five approaches from source code summarization and text summarization domains (i.e., BM25 [18], NMT [19], Hybrid-DeepCom [20], Transformer [21], and BART [22]) as the remaining baselines. Results of experimental study and human study show that `SOTitle` can generate higher-quality titles than these baselines.

To our best knowledge, the main contributions of our study can be summarized as follows.

- We propose an approach `SOTitle` to generate the title of the question post from Stack Overflow by considering both code snippet and problem description. Specifically, we combine the code snippet and problem description as the multi-modal input of the Transformer structure. Then we formalize post title generation for each programming language as separate but related tasks and utilize multi-task learning [15]. Finally, we use the SentencePiece method [16] to split the code snippet and the problem description to alleviate the OOV issue.
- We construct a high-quality corpus and this corpus contains 1,168,257 high-quality problem posts for four popular programming languages.
- We conduct empirical studies on our construct corpus to compare `SOTitle` with six state-of-the-art baselines, including the recent post title generation approach `Code2Que` [1]. Final empirical results show the competitiveness of `SOTitle`. Moreover, we verify the effectiveness of `SOTitle` via human study.
- We develop a browser plugin based on our proposed approach. By using this plugin, users can generate high-quality titles for their submitted question posts in Stack Overflow.
- We share our scripts, trained model, browser plugin, and corpora on our project homepage[1], which can facilitate the replication of our study and encourage more follow-up studies on the post title generation for Stack Overflow.

The rest of this paper is organized as follows. Section II analyzes related studies for post quality analysis, deep learning-based source code summarization, and deep learning-based text summarization. Section III introduces the framework of `SOTitle` and details of each component. Section IV and Section V show the experimental setup and results of empirical study and human study. Section VI discusses the main threats to the effectiveness of our empirical study. Section VII summarizes our study and shows several possible future directions.

II. Related Work

In this section, we first summarize the related work for post quality analysis on Stack Overflow. Since our approach aims to generate the post title by analyzing both the code snippet and the problem description, we also analyze the related work for deep learning-based source code summarization and text summarization. Finally, we emphasize the novelty of our study.

A. Post Quality Analysis on Stack Overflow

Improving post quality is an important research topic in Stack Overflow mining. For example, Correa and Sureka [10] investigated the closed questions in Stack Overflow and found that a good question should contain enough code for others to reproduce the problem. Nasehi et al. [23] performed qualitative assessment manually and investigated the important features of precise code examples in 163 Stack Overflow post answers. Yao et al. [12] found that the number of edits to a problem is a good indicator of the problem's quality. Arora et al. [9] proposed a new approach to improve the accuracy of the question quality prediction model by using the content extracted from similar historical question posts. Trienes et al. [11] studied the approach of identifying unclear problems in Stack Overflow. Ponzanelli et al. [24] developed an approach to automatically classify problems based on their quality. Gao et al. [1] were the first to use the sequence-to-sequence learning approach to automatically generate the post title based on the code snippet, which could improve the quality of the question post.

B. Deep Learning-based Text Summarization and Source Code Summarization

The problems close to our research problem are text summarization and source code summarization. In particular,

[1]https://github.com/NTDXYG/SOTitle

text summarization aims to summarize the text documents, which can obtain a brief overview of a large text document. While source code summarization can automatically generate the corresponding code comments by analyzing the semantic information of the target code, which can help developers understand the design purpose and functionality of the code. Recently, the deep learning-based method is the popular research direction of these two research problems. In this subsection, we mainly analyze the related work for these two problems.

For deep learning-based source code summarization, Iyer et al. [25] first studied the problem of source code summarization problem and proposed the method CODE-NN. This method mainly uses LSTMs in the encoder and decoder. At the same time, it used an attention mechanism, which can assign higher weights to related word elements in the sequence. Hu et al. [26] proposed the method DeepCom. Then they further extended the method DeepCom and proposed the method Hybrid-DeepCom [20], which combines the lexical information and grammatical information of the code, and splits the identifiers based on the camel case naming convention to alleviate the OOV problem. Finally, they used beam search to improve comment quality. Yang et al. [27] proposed a novel method ComFormer based on Transformer and fusion method-based hybrid code presentation LeClair et al. [28] considered the graph neural network, based on the graph2seq model [29]. Ahmad et al. [30] considered the Vanilla-Transformer architecture, and used a relative position representation and copy mechanism. Feng et al. [31] proposed the bi-modal pre-training model CodeBERT based on Transformer neural architecture for programming language and natural language. Wang et al. [32] proposed a BERT-based functional enhanced transformer model, they proposed a new enhancer to generate higher-quality code summarization.

For deep learning-based text summarization, Rush et al. [33] proposed a fully data-driven approach to abstracting sentence summaries. This approach used a local attention-based model to generate each word of the summary conditioned on the input sentence. See et al. [34] introduced a pointer network to address the problem that seq2seq models often do not accurately reproduce factual details. The approach can both generate words from a vocabulary through generators and copy content from a source through pointers. Liu and Lapata [35] used a pre-trained Bert [36] as a sentence encoder and a Transformer as a document encoder. The classifier of sentence representation is used for sentence selection. It used the knowledge of fine-tuned BERT to generate better text summaries. Instead of only pre-training the encoder, the Bart model proposed by Lewis et al. [22] jointly pre-trained a seq2seq model that combines a bidirectional encoder and an autoregressive decoder.

C. Novelty of Our Study

Most relevant to our study is the approach `Code2Que` proposed by Gao et al. [1]. `Code2Que` [1] is an LSTM-based sequence-to-sequence deep learning model that helps improve the post quality by automatically generating post titles based on a given code snippet. However, `Code2Que` only considers a single input modality and ignores the problem description of the question post, which can provide valuable information for title generation that code snippets sometimes cannot provide. Therefore, our approach `SOTitle` extracts useful information from both modalities (i.e., the problem description and the code snippet) simultaneously to generate high-quality titles for question posts. In addition, we formalize the generation of question post titles for each programming language as separate but related tasks, which can allow related tasks to improve each other's performance by using shared and complementary information.

III. Our Proposed Approach

The overall framework of our approach is shown in Fig. 2. In this figure, we can find that `SOTitle` consists of three phases (i.e., corpus construction, model construction, and model application). In particular, (1) in the corpus construction phase, we design three heuristic rules to collect high-quality question posts from Stack Overflow. In our constructed corpus, we mainly focus on posts related to four programming languages and extract the code snippet, the problem description, and the title as a triplet from each post. More details of corpus construction can be found in Section IV-A. (2) In the model construction, we first model the multi-modal input by concatenating the code snippet and the problem description. Then we use the SentencePiece method [16] to split these two modalities to alleviate the OOV problem. Later we formalize question post title generation for each programming language as separate but related tasks and resort to multi-task learning. Finally, our model is fine-tuned based on a pre-trained Transformer model T5 [37]. (3) In the model application phase, for a new question post, we input its code snippet and problem description to the constructed model. Then the trained model can automatically generate the corresponding post title through the beam search algorithm. In the rest of this section, we show the details of the model construction phase.

A. Multi-modal Input Modeling

Since our study needs to deal with different programming languages simultaneously, we should alleviate the OOV problem. In this study, we adopt the SentencePiece method [16] to handle our input X. The SentencePiece method considers the input sequence as a Unicode encoded sequence, thus this method does not depend on the language representation, which helps to handle different programming languages.

Different from previous studies, we want to extract information from both modalities (i.e., code snippet and problem descriptions) simultaneously. In this study, we model the multi-modal input by concatenating the code snippet and the problem description together. Specifically, we concatenate the code snippet sequence X_{code} and the problem description sequence X_{desc} via a special identifier ($<\text{code}>$) to distinguish X_{code} and X_{desc}. As shown in Fig. 2, since we considered generating titles for different programming language problem posts as

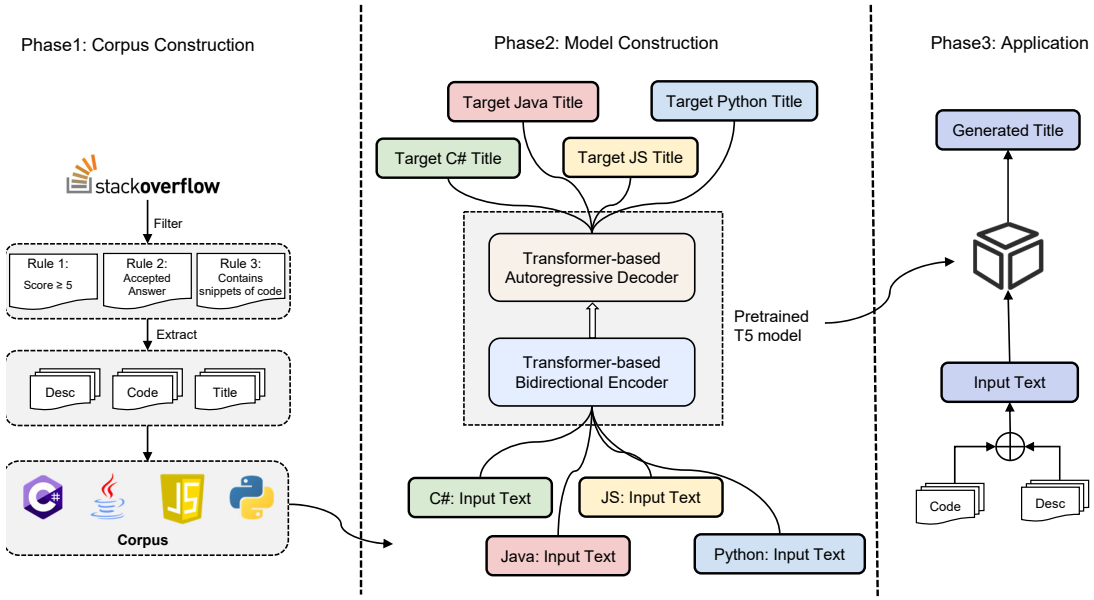

Fig. 2. Framework of our proposed approach `SOTitle`

separate but related tasks, we prefixed the input X of each programming language with a task-specific prefix (e.g., the prefix "JS:" denotes JavaScript) to make the model distinguish between the different tasks. The format of the input is shown as follows.

$$\mathrm{X} = \text{ prefix } \oplus X_{\text{desc}} \oplus < \text{code} > \oplus X_{\text{code}} \tag{1}$$

B. Transformer-based Bidirectional Encoder

The encoder of our model aims to learn the representation of the problem posts $X = (x_1, x_2, \cdots, x_m)$. The encoder is composed of a stack of "blocks", each of which consists of two subcomponents: a self-attention layer, followed by a feed-forward network.

Self-attention [38] is calculated based on queries (Q), keys (K), and values (V). The dot product between the queries and keys is first calculated, then each is divided by $\sqrt{d_k}$ and the softmax function is applied to get the weight of the corresponding value.

$$\text{Attention } (Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right) V \tag{2}$$

The feed-forward neural network (FFN) consists of two linear transformations, with a nonlinear transformation provided through the Relu activation.

$$\text{FFN}(x) = \max\left(0, xW_1 + b_1\right) W_2 + b_2 \tag{3}$$

Layer normalization [39] is applied to the inputs of each child component. After layer normalization, a residual skip connection [40] adds the inputs of each child component to its output. Dropout [41] is applied within the feed-forward network, on the skip connection, on the attention weights, and at the input and output of the entire stack.

Note that the encoder uses all sub-tokens in the input sequence for learning so that the learned representation of each sub-token contains information about the entire input sequence. Since we encode the information of all the modalities in a sequence, the learned representation of each sub-token contains information about the other modalities.

C. Transformer-based Autoregressive Decoder

The structure of the decoder is similar to the encoder. The difference is that it uses a standard attention mechanism to focus on the encoder output after each self-attentive layer. The self-attention mechanism in the decoder also uses a type of autoregressive or causal self-attention that allows the model to only focus on the past outputs.

D. Model Fine-tuning Process

Our model is fine-tuned based on a pre-trained Transformer language model T5 [37]. During the fine-tuning process, we do not tune the parameters of the model's bias and LayerNorm.weight's weights, and then use the Adafactor optimizer [42] to fine-tune the other parameters.

The input text x is tokenized as $\left\{x_1, \cdots, x_{|x|}\right\}$ and encoded as the learned embedding $e^x = \left\{e_1^x, \cdots, e_{|x|}^x\right\}$. The encoder takes e^x as the input and outputs a joint representation of their context $h = \left\{h_1^x, \ldots, h_{|x|}^x\right\} = \text{Enc}\left(e^x\right)$. The decoder then iterates on the previously generated token $y_{<j}$ via self-attention, the encoder outputs h via cross-attention, and then predicts the probability of the next text token $P_\theta\left(y_j \mid y_{<j}, x\right) = \text{Dec}\left(y_{<j}, h\right)$.

We train our model parameters θ by minimizing the negative log-likelihood of the target text tokens y for a given input text x. For each task, the formula can be defined as follows.

$$\mathcal{L}_{\theta}^{\text{Task}} = -\sum_{j=1}^{|y|} \log P_{\theta}\left(y_j \mid y_{<j}, x\right) \tag{4}$$

Finally, the loss functions for these four different programming languages (i.e., Java, C#, Python, and JavaScript) can be defined as follows.

$$\mathcal{L}_{\theta} = (\sum_{i=1}^{4} \mathcal{L}_{\theta}^{\text{Task}_i})/4 \tag{5}$$

IV. Experiment Setup

In our empirical study, we want to answer the following three research problems (RQs):

RQ1: Can our proposed approach `SOTitle` outperform state-of-the-art baselines via automatic evaluation?

RQ2: What are the contribution of different input modalities for the performance of `SOTitle`?

RQ3: Can our proposed approach `SOTitle` outperform the state-of-the-art post title generation approach `Code2Que` via human study?

In RQ1 and RQ3, we aim to show the competitiveness of `SOTitle` via automatic evaluation and human study. In RQ2, we aim to analyze the effect of the multi-modal input modeling for `SOTitle`.

A. Experimental Subject

We select question posts from Stack Overflow as our experimental subject. Fig. 3 shows a question post for the Java programming language. This post contains a short post title, the problem description with the corresponding code snippet, one or more relevant answers (one of which was marked as accepted), and multiple tags.

In this study, we mainly select question posts for four popular programming languages (i.e., Java, C#, Python, and JavaScript). Specifically, we first use Java, C#, Python, and JavaScript tags to collect related question posts. To improve the quality of the gathered question posts, we propose three heuristic rules based on our manual analysis and suggestions from previous studies [43] [44].

- **Rule 1:** The score of the question posts is not smaller than 5.
- **Rule 2:** The question posts should have the accepted answers.
- **Rule 3:** The question posts should contain the code snippets.

After using these three rules, we find that the percentage of selected posts in Stack Overflow[2] is only 6%, which means we finally select 1,168,257 posts from 20,511,138 posts. Then we extract the problem description, the code snippet, and the post title as the triplet ⟨Description, Code, Title⟩ and add this

[2]https://archive.org/download/stackexchange, downloaded in October 2020

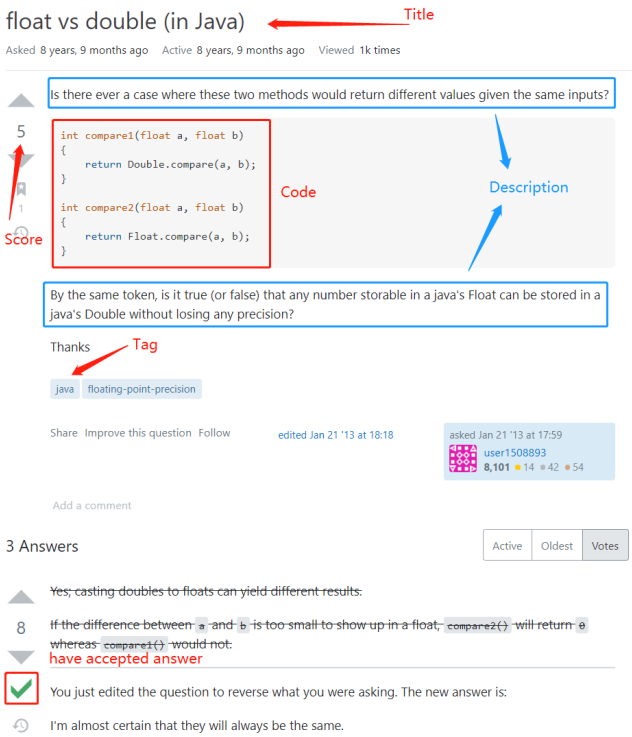

Fig. 3. A question post for Java programming language from Stack Overflow

triplet to our corpus. Finally, there are 68,959 posts for Java, 71,817 posts for C#, 72,742 posts for Python, and 70,780 posts for JavaScript. Then we randomly selected 60,000 posts as the training set, 5,000 posts as the testing set, and the remaining posts as the validation set for each programming language. Detailed statistical information of corpus split results can be found in Table I. Moreover, we also show the length statistics of code snippet, problem description, and title in Table II.

TABLE I
STATISTICAL INFORMATION OF CORPUS SPLIT RESULTS

Language	Training	Validation	Testing
Java	60,000	3,959	5,000
C#	60,000	6,817	5,000
Python	60,000	7,742	5,000
JavaScript	60,000	5,780	5,000

B. Performance Measures

In our study, we aim to show the competitiveness of `SOTitle` via both automatic evaluation and human evaluation. In this section, we first introduce the performance measure used in the automatic evaluation. Then we introduce the details of our human study methodology.

1) Automatic Evaluation: Rouge [17] is a recall-based measure, which is used to calculate the lexical overlap between machine-generated summaries and reference summaries. This

TABLE II
LENGTH STATISTICS OF CODE SNIPPET, PROBLEM DESCRIPTION, AND TITLE

Language	Code Length				Description Length				Title Length			
	Average	Mode	Median	<256	Average	Mode	Median	<256	Average	Mode	Median	<16
Java	173.98	20	84	82.31%	114.09	58	89	93.45%	9.42	7	9	92.30%
C#	128.54	16	73	88.40%	120.15	62	93	92.35%	9.69	8	9	90.90%
Python	124.98	24	70	89.11%	106.55	48	85	95.03%	9.41	8	9	92.86%
JavaScript	142.83	20	82	86.68%	103.96	61	83	95.34%	9.33	8	9	93.15%

Problem Description:
Assuming I have a column called df.Text which contains text (more that 1 sentence) and I want to use polyglot Detector to detect the language and store the value in a new column df['Text-Lang'] how do I ensure I also capture the other details like code and confidence ...
Code Snippet:
testEng ="This is English"
lang = Detector(testEng)
print(lang.language)
...

Title 1: How to detect the language and store the value in a pandas dataframe ?	**Title 2:** Polyglot Detector Detector Detector Detector
Q1-1: What is the score of Title 1 ? A. EXCELLENT. B. GOOD. C. BORDERLINE. D. BAD.	**Q2-1: What is the score of Title 2 ?** A. EXCELLENT. B. GOOD. C. BORDERLINE. D. BAD.
Q2-1: What do you think of Title 1 ? (multiple choices) A. Relevance GOOD. B. Conciseness GOOD. C. Expressiveness GOOD.	**Q2-2: What do you think of Title 2 ? (multiple choices)** A. Relevance BAD. B. Conciseness BAD. C. Expressiveness BAD.

Fig. 4. A survey example used in our human study

performance measure has been successfully used in previous source code summarization and text summarization studies [37], [45], [46]. In particular, Rouge-1 and Rouge-2 are based on unigram and bigram respectively. Rouge-L is based on the LCS (Longest Common Subsequence). In our study, we use Rouge[3] to calculate the Rogue value.

2) Human Evaluation: In our human study, we mainly follow the methodology of Chen et al. [47]. Specifically, we recruited six master students with more than five years on project development and familiar with the usage of Stack Overflow. Before conducting the human study, we provide the guidelines for scoring the post title quality. Since `Code2Que` [1] is the state-of-the-art baseline for post title generation, we mainly compare the quality of titles generated by `SOTitle` and `Code2Que`.

Then we randomly select 50 question posts for each programming language. Given a question post, we use `Code2Que` and `SOTitle` to generate titles respectively. In summary, we have 200 titles generated by `Code2Que`, which can be represented by $T^i = \{t_1^i, t_2^i, \cdots, t_{200}^i\}$), and 200 titles generated by `SOTitle`, which were represented by $T^j = \left\{t_1^j, t_2^j, \cdots, t_{200}^j\right\}$.

We randomly ordered the selected question posts, and the students did not know which title was generated by our approach. We use the body of k-th question post and the associated t_k^i and t_k^j and the two generated titles are shown in random order. We asked students to carefully review each question post. First, we asked them to rate each of the assigned titles, the meaning and the corresponding score as shown as follows:

- **Excellent (2).** This title can be used as the post title without modification.
- **Good (1).** This title can reflect the main idea of the question post, but has some problems (such as incompleteness, repetition, or grammatical problem). Therefore, this title can be used as the post title with minor modifications.
- **Borderline (0).** This title can reflect the idea of the question post, but it lacks necessary details or is confusing. Therefore, this title can be used as the post with major modifications.
- **Bad (-1).** This title cannot reflect the idea of the question post. Therefore, this title needs to be rewritten.

By comparing the final score of T^i and T^j, we can verify whether `SOTitle` can generate higher-quality titles than `Code2Que`.

Second, we asked the hired students to give reasons to support their scores. To simplify this process, we only surveyed the advantages of the generated titles which are scored as "Excellent" or "Good" and the disadvantages of the generated titles which are scored as "Borderline" or "Bad". The reasons are summarized from three perspectives.

- **Relevance.** The generated title can capture the main idea of the question post with correct and sufficient detail.
- **Conciseness.** The generated title does not contain unnecessary information and uses a short sentence to show the main idea of the post.
- **Expressiveness.** the generated title can be clearly described.

Here we use an example post shown in Fig. 4 from Stack Overflow[4] to show the investigation process. With different

[3]https://github.com/pltrdy/rouge

[4]URL: https://stackoverflow.com/questions/45958129

answers in Q1, the options in Q2 are slightly different: (1) When the answer of Q1 is "Excellent" or "Good", the options of Q2 will ask the students why it is good, such as Q2-1. (2) When the answer of Q1 is "Borderline" or "Bad", the options of Q2 will ask students why it is bad, such as Q2-2.

C. Implementation Details

In our empirical study, we use Transformers[5] to implement our proposed approach `SOTitle`. The word embedding dimensions and hidden sizes are set to 768, and the number of attention heads and layers is set to 12. All parameters were optimized with Adafactor [42], and the initial learning rate is set to 0.0005. During training, the batch size is set to 30. The maximum length of the encoder and decoder is set to 512 and 30 respectively. To alleviate the overfitting problem, we adopted a Dropout mechanism and set the loss rate to 0.1. Finally, we also use the early stop method [48] to further alleviate the overfitting problem, and the weights with the highest performance on the validation set is taken as the final parameter value of the neural network.

We run all the experiments on a computer with an Inter(R) Xeon(R) Silver 4210 CPU and a GeForce RTX3090 GPU with 24 GB memory. The running OS platform is Windows OS.

D. Baselines

We select six state-of-the-art baselines to show the effectiveness of our proposed approach. We first select a recent post title generation approach `Code2Que` as the first baselines.

Code2Que. `Code2Que` [1] is the state-of-the-art baseline for post title generation. `Code2Que` automatically generates titles for question posts through an LSTM-based deep learning approach.

Then we select three baselines from the text summarization domain.

BM25. `BM25` [18] is a bag-of-word retrieval function, which can be used to estimate the relevance of documents for a given search query. This is the information retrieval-based baseline.

Transformer. `Transformer` [21] mode improves the sequence processing ability through the attention mechanism. Recently, Transformer has been successfully used in different NLP understanding and generation tasks.

BART. `BART` [22] pre-trained the Transformer model by combining bidirectional encoder and autoregressive decoder. The BART model performs well on text summarization tasks. In our study, we used the `BART` model and then fine-tuned this model for our task.

Finally, we select two baselines from source code summarization.

NMT. Jiang et al. [19] used Neural machine translation (NMT) technology to automatically "translate" code changes into commit messages. We choose `NMT` as a baseline since it can achieve promising performance in generating commit messages for the code changes.

Hybrid-DeepCom. `Hybrid-DeepCom` [20] is a neural model used for code comment generation, which learns syntactic and semantic information about the code through two different encoders, respectively. It should be noticed that the structure of both `NMT` and `Hybrid-DeepCom` is the same except for the encoder part. We use the model `Hybrid-DeepCom` to generate post titles by learning code snippet and problem description separately through two different encoders.

To make a fair comparison, we slightly adapted these baselines to deal with two modalities. In particular, for the baseline `Hybrid-DeepCom`, we feed the two modalities into two encoders separately according to their original approach description. For the remaining baselines, we concatenate the code snippet and the problem description together and then feed them into one encoder.

[5]https://github.com/huggingface/transformers

E. Browser Plug-in Support

To facilitate the use of our proposed approach, we developed a browser plug-in based on our constructed model. Fig. 5 shows the screenshot of our developed browser plug-in.

V. Result Analysis

A. Result Analysis for RQ1

RQ1: Can our proposed approach `SOTitle` outperform state-of-the-art baselines via automatic evaluation?

The comparison results between our proposed approach `SOTitle` and baselines can be found in Table III. For each column, we emphasize the best value in bold. It can be found that our proposed approach `SOTitle` can significantly outperform six baselines in terms of the performance measure Rouge for different programming languages.

In particular, for our considered baselines, we find deep learning-based baselines (i.e., `NMT`, `Hybrid-DeepCom`, `Code2Que`, `Transformer`, `BART`, `SOTitle`) can achieve better performance than information retrieval-based baseline (i.e., `BM25`). The information retrieval-based baseline aims to retrieve historical similar posts based on similarity, and the performance of this kind of baseline depends on whether similar posts can be found. In contrast, deep learning-based approaches can learn semantic information from two different modalities.

In terms of Rouge performance measures, our approach can significantly outperform the post title generation baseline `Code2Que` [1]. For example, in terms of Rouge-L, compared to `Code2Que`, `SOTitle` can improve the performance by 6.187%, 6.005%, 5.029%, and 5.955% for Java, Python, C#, and JavaScript respectively. The potential reason is that we consider the post title generation problem for different programming languages as different but related tasks, which can improve each other's performance by sharing information and complementing each other.

Except for the Rouge performance measures, we also compare `SOTitle` with baselines in terms of METEOR [49] and BLEU [50] performance measures. Final results also show the competitive performance of `SOTitle`. Limited by the length of the paper, we show the detailed results on our project homepage.

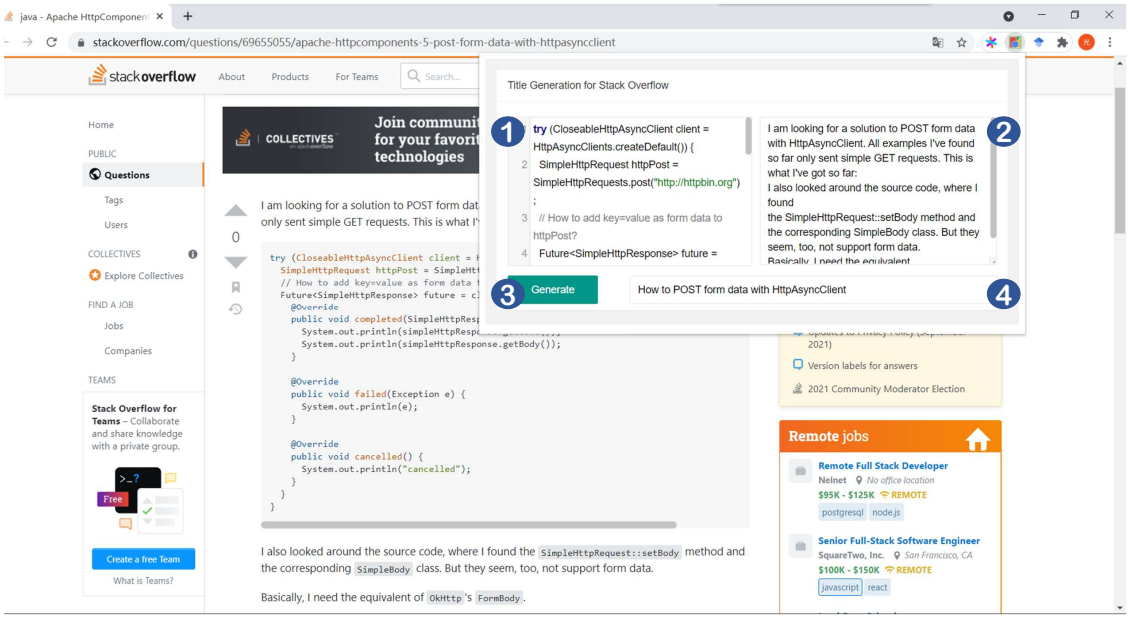

Fig. 5. Screenshot of our developed browser plug-in. When using this plug-in, the developer can first copy the code snippet in ① and the detailed problem description in ②. Then the developer can click the generate button in ③. Finally, the generated title can be displayed in ④.

TABLE III
COMPARISON RESULTS BETWEEN OUR PROPOSED APPROACH AND STATE-OF-THE-ART BASELINES

Approach	Java			C#			Python			JavaScript		
	Rouge-1	Rouge-2	Rouge-L	Rouge-1	Rouge-2	Rouge-L	Rouge-1	Rouge-2	Rouge-L	Rouge-1	Rouge-2	Rouge-L
BM25	10.045	1.150	9.294	10.183	1.760	9.552	11.443	1.475	10.487	9.782	1.182	9.205
NMT	18.017	3.393	17.166	18.656	4.885	17.850	19.482	4.090	18.549	17.341	3.279	16.738
Transformer	12.689	1.957	12.332	12.252	2.046	12.115	14.125	2.236	13.602	13.153	2.082	12.751
BART	24.801	8.918	23.343	23.395	9.545	22.286	27.272	10.321	25.672	25.421	9.724	24.095
Hybrid-DeepCom	10.620	1.511	10.232	12.346	2.684	11.884	11.512	1.853	11.156	14.103	2.421	13.323
`Code2Que`	22.021	6.787	21.075	23.476	8.605	22.584	24.635	8.301	23.282	23.958	8.164	22.942
`SOTitle`	**29.328**	**10.968**	**27.262**	**29.550**	**11.958**	**27.613**	**31.828**	**11.985**	**29.287**	**31.150**	**11.822**	**28.897**

Fig. 6 shows the titles generated by `SOTitle` and baselines for a Java question post[6]. In this example, we can find that the title generated by `BM25` is not relevant to the question post. The titles generated by `NMT`, `Hybrid-DeepCom`. `Transformer` and `BART` cannot correctly express the core idea of the question post. `Code2Que` fails to capture the details of the problem. However, the title generated by our approach can accurately and fluently express the key information in this post.

Summary for RQ1: `SOTitle` can achieve better performance than six state-of-the-art baselines for different programming languages via automatic evaluation.

[6]URL: https://stackoverflow.com/questions/52852143

B. Result Analysis for RQ2

RQ2: What are the contribution of different input modalities for the performance of `SOTitle`?

In this RQ, we aim to investigate the contribution of different input modalities to the performance of `SOTitle`. Moreover, we also investigate the contribution of different input modalities to the post title generation baseline `Code2Que`, since `Code2Que` only considers the code snippet in the original study. Here we use different subscripts to distinguish different control approaches and the meaning of these subscripts is shown as follows.

- **code.** The corresponding control approach only uses the code snippet as the input modal.
- **desc.** The corresponding control approach only uses the problem description as the input modal.

Table IV shows the performance of `SOTitle` and `Code2Que` with different combinations of input modes. Here $\texttt{Code2Que}_{\text{code}}$ can denote the performance of `Code2Que`'s

Problem body :	Title :

I see that I can append to a list while iterating over it
`lst = [1]`
`for i in lst:`
`  lst.append(i+1)`
`  print(i)`
Am I allowed to make use of this behavior? or is it discouraged? I note that the same can not be said for `set`
`lst = set([1])`
`for i in lst:`
`  lst.add(i+1)`
`  print(i)`
> Error: size changed during iteration.
	Ground truth: Is it correct to append to a list while iterating over it ?
	BM25: Iterating over partitions in Python
	NMT: How to append to a list ?
	Hybrid-DeepCom: Appending a list to a list comprehension in python
	Transformer: Python list comprehension
	BART: Python list append ( )
	Code2Que: How to append to a list in python ?
	SOTitle: Is it discouraged to append to a list while iterating over it ?

Fig. 6. The titles generated by SOTitle and baselines for a question post related to the Java programming language

TABLE IV
THE PERFORMANCE OF SOTITLE AND CODE2QUE WITH DIFFERENT COMBINATIONS OF INPUT MODES

Approach	Java			C#			Python			JavaScript		
	Rouge-1	Rouge-2	Rouge-L	Rouge-1	Rouge-2	Rouge-L	Rouge-1	Rouge-2	Rouge-L	Rouge-1	Rouge-2	Rouge-L
$\texttt{Code2Que}_{\text{code}}$	14.157	2.576	13.590	14.895	3.388	13.602	12.705	2.184	12.015	12.648	2.109	12.112
$\texttt{SOTitle}_{\text{code}}$	15.064	2.644	14.255	15.445	3.525	14.255	13.534	2.236	12.801	13.467	2.295	12.952
$\texttt{Code2Que}_{\text{desc}}$	21.213	6.042	20.637	22.896	8.022	21.957	24.012	8.001	22.439	23.288	7.979	22.310
$\texttt{SOTitle}_{\text{desc}}$	24.586	9.101	23.083	24.812	10.063	23.439	26.966	10.408	25.226	26.294	10.176	24.922
Code2Que	22.021	6.787	21.075	23.476	8.605	22.584	24.635	8.301	23.282	23.958	8.164	22.942
SOTitle	**29.328**	**10.968**	**27.262**	**29.550**	**11.958**	**27.613**	**31.828**	**11.985**	**29.287**	**31.150**	**11.822**	**28.897**

original study. In terms of Rouge-L performance measure, when we only use the code snippets as the input, the Rouge-L scores of SOTitle are 14.255, 14.255, 12.801, and 12.952 for the Java, C#, Python, and JavaScript respectively; the Rouge-L scores of Code2Que are 13.590, 13.602, 12.015, and 12.112 respectively. When we only use the problem description as the input, compared to only using the code snippets as the input. The Rouge-L scores of SOTitle can be improved by 61.9%, 64.2%, 97.1%, and 92.4% respectively; the Rouge-L scores of Code2Que can be improved by 51.9%, 61.2%, 86.8%, and 84.2% respectively. This shows that for the post title generation task, more useful information can be extracted from the problem description. When we use both the problem description and the code snippet as the input, the Rouge-L scores of SOTitle can be improved by 91.2%, 93.7%, 128.8%, and 123.1% respectively. The Rouge-L scores of Code2Que can be improved by 55.1%, 66.0%, 86.8%, and 89.4%, respectively. This shows that the information from the code snippet and the problem description is complimentary and both two modalities should be considered for post title generation.

Summary for RQ2: Problem description can provide more valuable information than the code snippet for post title generation. Moreover, considering two modalities together can improve the performance of SOTitle and Code2Que.

C. Result Analysis for RQ3

RQ3: Can our proposed approach SOTitle outperform the state-of-the-art post title generation baseline Code2Que via human study?

According to the methodology of the human study introduced in Section IV, we randomly select 50 question posts for each programming language. For each post, two titles are generated by Code2Que and SOTitle respectively. Each question post was evaluated separately by six master students during the survey. Fig. 7 and Fig. 8 show the statistical results of 200 feedback comments, each containing four perspectives: Overall Rating, Relevance, Conciseness, and Expressiveness.

As shown in Fig. 7, the left and right subfigures show the votes for the titles generated by Code2Que and the titles generated by SOTitle, respectively. We notice that 81% of

the human ratings for the titles generated by `SOTitle` are not less than 1, which means they are considered to have good quality and can be used as titles without major modifications. Then, we use Fleiss Kappa [51] to measure the agreement among these six students. The overall Kappa value based on the comparison results between `SOTitle` and `Code2Que` is 0.766, which indicates substantial agreement among these students. Moreover, the distribution of human comments (in Fig. 8) shows that the titles generated by `SOTitle` received more positive comments after compared to the titles generated by `Code2Que` in terms of three perspectives (i.e., Relevance, Conciseness, and Expressiveness). Therefore, our proposed approach can help to improve the post title quality. In our human study, we also find some negative votes for both approaches. After manual analysis, we find these approaches cannot generate useful titles when the code snippets are too complex or the problem descriptions lack enough useful information. We show the examples with negative votes and their corresponding reasons in our homepage.

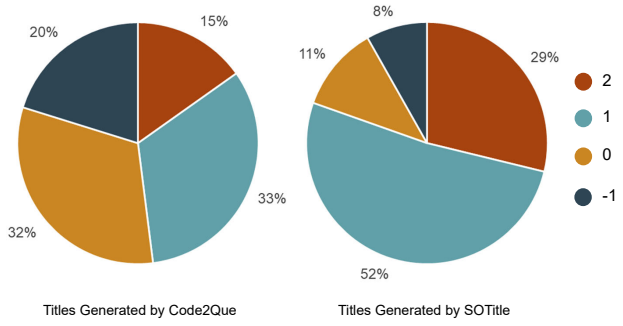

Fig. 7. Rating distribution of human votes and each sector presents the proportion of votes with the corresponding rating score.

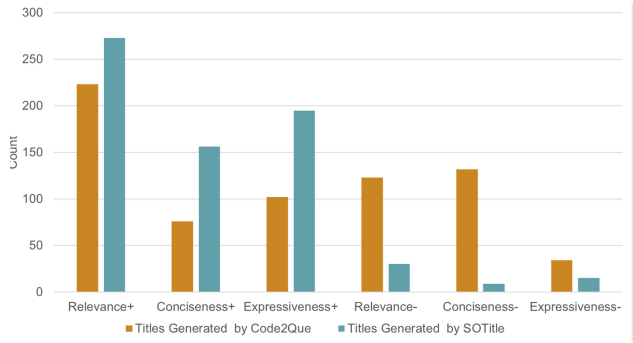

Fig. 8. Advantages (+) and disadvantages (-) distribution of human comments. Each bar presents the count of corresponding perspective.

Summary for RQ3: `SOTitle` can generate higher-quality titles than the baseline `Code2Que` via human study.

VI. Threats to Validity

In this section, we mainly analyze the potential threats to the validity of our empirical study.

Internal Threats. The first threat to internal validity concerns potential faults in the implementation of our proposed approach and baselines. To alleviate this threat, we use mature frameworks and use software testing to guarantee the code quality. The second threat to internal validity is the method used to split the tokens. In our study, we consider the SentencePiece method. This method is a language-independent subword tokenizer and detokenizer, and its effectiveness has been verified in the neural machine translation problem [16].

External Threats. The threat to external validity concerns the quality and generalization ability of our constructed corpus. To alleviate this threat, we mainly consider question posts for four popular programming languages and consider three heuristic rules to identify and remove low-quality posts.

Construct Threats. The threat to construct validity comes from human studies, which may introduce bias. To ensure that all students can correctly understand our questionnaire, we provided a tutorial before our human study.

VII. Conclusion

In this study, we propose the Transformer-based approach `SOTitle` for automatic post title generation by leveraging the code snippets and the problem description (i.e., the multi-modal input) in the question post. To verify the effectiveness of our proposed approach, we gathered 1,168,257 high-quality problem posts for four popular programming languages from Stack Overflow. Experimental results show the competitiveness of our proposed approach after comparing the six state-of-the-art baselines in terms of Rouge performance measures. Moreover, we also conduct a human study to verify the effectiveness of `SOTitle` after comparing the post title generation approach `Code2Que`.

In the future, we first aim to further improve the performance of `SOTitle` by considering more advanced deep learning methods (such as graph neural networks for code snippets). We second aim to combine `SOTitle` with other kinds of approaches (such as information retrieval-based approaches and template-based approaches).

Acknowledgment

The authors would like to thank the anonymous reviewers for their insightful comments and suggestions, which can substantially improve the quality of this work. Ke Liu and Guang Yang have contributed equally to this work and they are co-first authors. This work is supported in part by the National Natural Science Foundation of China (Grant no. 61872263), The Open Project of State Key Laboratory of Information Security (Institute of Information Engineering, Chinese Academy of Sciences) (Grant No. 2020-MS-07).

REFERENCES

[1] Z. Gao, X. Xia, J. Grundy, D. Lo, and Y.-F. Li, "Generating question titles for stack overflow from mined code snippets," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 29, no. 4, pp. 1–37, 2020.

[2] K. Cao, C. Chen, S. Baltes, C. Treude, and X. Chen, "Automated query reformulation for efficient search based on query logs from stack overflow," in *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 2021, pp. 1273–1285.

[3] J. Liu, S. Baltes, C. Treude, D. Lo, Y. Zhang, and X. Xia, "Characterizing search activities on stack overflow," in *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. Association for Computing Machinery, 2021, p. 919–931.

[4] X. Chen, C. Chen, D. Zhang, and Z. Xing, "Sethesaurus: Wordnet in software engineering," *IEEE Transactions on Software Engineering*, vol. 47, no. 9, pp. 1960–1979, 2021.

[5] M. Asaduzzaman, A. S. Mashiyat, C. K. Roy, and K. A. Schneider, "Answering questions about unanswered questions of stack overflow," in *2013 10th Working Conference on Mining Software Repositories (MSR)*. IEEE, 2013, pp. 97–100.

[6] A. Anderson, D. Huttenlocher, J. Kleinberg, and J. Leskovec, "Discovering value from community activity on focused question answering sites: a case study of stack overflow," in *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2012, pp. 850–858.

[7] X. Jin and F. Servant, "What edits are done on the highly answered questions in stack overflow? an empirical study," in *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*. IEEE, 2019, pp. 225–229.

[8] M. Duijn, A. Kucera, and A. Bacchelli, "Quality questions need quality code: Classifying code fragments on stack overflow," in *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*. IEEE, 2015, pp. 410–413.

[9] P. Arora, D. Ganguly, and 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.

[10] D. Correa and 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.

[11] J. Trienes and K. Balog, "Identifying unclear questions in community question answering websites," in *European Conference on Information Retrieval*. Springer, 2019, pp. 276–289.

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

[13] T. Zhang, G. Upadhyaya, A. Reinhardt, H. Rajan, and M. Kim, "Are code examples on an online q&a forum reliable?: a study of api misuse on stack overflow," in *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)*. IEEE, 2018, pp. 886–896.

[14] L. Tóth, B. Nagy, D. Janthó, L. Vidács, and T. Gyimóthy, "Towards an accurate prediction of the question quality on stack overflow using a deep-learning-based nlp approach." in *ICSOFT*, 2019, pp. 631–639.

[15] Y. Zhang and Q. Yang, "A survey on multi-task learning," *IEEE Transactions on Knowledge and Data Engineering*, pp. 1–1, 2021.

[16] T. Kudo and J. Richardson, "Sentencepiece: A simple and language independent subword tokenizer and detokenizer for neural text processing," 2018.

[17] C.-Y. Lin, "Rouge: A package for automatic evaluation of summaries," in *Text summarization branches out*, 2004, pp. 74–81.

[18] S. Robertson and H. Zaragoza, *The probabilistic relevance framework: BM25 and beyond*. Now Publishers Inc, 2009.

[19] S. Jiang, A. Armaly, and C. McMillan, "Automatically generating commit messages from diffs using neural machine translation," in *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2017, pp. 135–146.

[20] X. Hu, G. Li, X. Xia, D. Lo, and Z. Jin, "Deep code comment generation with hybrid lexical and syntactical information," *Empirical Software Engineering*, vol. 25, no. 3, pp. 2179–2217, 2020.

[21] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in neural information processing systems*, 2017, pp. 5998–6008.

[22] M. Lewis, Y. Liu, N. Goyal, M. Ghazvininejad, A. Mohamed, O. Levy, V. Stoyanov, and L. Zettlemoyer, "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*, 2020, pp. 7871–7880.

[23] S. M. Nasehi, J. Sillito, F. Maurer, and C. Burns, "What makes a good code example?: A study of programming q&a in stackoverflow," in *2012 28th IEEE International Conference on Software Maintenance (ICSM)*. IEEE, 2012, pp. 25–34.

[24] L. Ponzanelli, A. Mocci, A. Bacchelli, and M. Lanza, "Understanding and classifying the quality of technical forum questions," in *2014 14th International Conference on Quality Software*. IEEE, 2014, pp. 343–352.

[25] S. Iyer, I. Konstas, A. Cheung, and L. Zettlemoyer, "Summarizing source code using a neural attention model," in *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2016, pp. 2073–2083.

[26] X. Hu, G. Li, X. Xia, D. Lo, and Z. Jin, "Deep code comment generation," in *2018 IEEE/ACM 26th International Conference on Program Comprehension (ICPC)*. IEEE, 2018, pp. 200–20 010.

[27] G. Yang, X. Chen, J. Cao, S. Xu, Z. Cui, C. Yu, and K. Liu, "Comformer: Code comment generation via transformer and fusion method-based hybrid code representation," in *2021 8th International Conference on Dependable Systems and Their Applications (DSA)*. IEEE, 2021, pp. 30–41.

[28] A. LeClair, S. Haque, L. Wu, and C. McMillan, "Improved code summarization via a graph neural network," in *Proceedings of the 28th International Conference on Program Comprehension*, 2020, pp. 184–195.

[29] K. Xu, L. Wu, Z. Wang, Y. Feng, M. Witbrock, and V. Sheinin, "Graph2seq: Graph to sequence learning with attention-based neural networks," *arXiv preprint arXiv:1804.00823*, 2018.

[30] W. Ahmad, S. Chakraborty, B. Ray, and K.-W. Chang, "A transformer-based approach for source code summarization," in *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, 2020, pp. 4998–5007.

[31] Z. Feng, D. Guo, D. Tang, N. Duan, X. Feng, M. Gong, L. Shou, B. Qin, T. Liu, D. Jiang *et al.*, "Codebert: A pre-trained model for programming and natural languages," in *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: Findings*, 2020, pp. 1536–1547.

[32] R. Wang, H. Zhang, G. Lu, L. Lyu, and C. Lyu, "Fret: Functional reinforced transformer with bert for code summarization," *IEEE Access*, vol. 8, pp. 135 591–135 604, 2020.

[33] A. M. Rush, S. Chopra, and J. Weston, "A neural attention model for abstractive sentence summarization," in *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, 2015, pp. 379–389.

[34] A. See, P. J. Liu, and C. D. Manning, "Get to the point: Summarization with pointer-generator networks," pp. 1073–1083, 2017.

[35] Y. Liu and M. Lapata, "Text summarization with pretrained encoders," in *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, 2019, pp. 3730–3740.

[36] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," in *NAACL-HLT (1)*, 2019.

[37] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu, "Exploring the limits of transfer learning with a unified text-to-text transformer," *Journal of Machine Learning Research*, vol. 21, pp. 1–67, 2020.

[38] J. Cheng, L. Dong, and M. Lapata, "Long short-term memory-networks for machine reading," in *2016 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 2016, pp. 551–561.

[39] J. L. Ba, J. R. Kiros, and G. E. Hinton, "Layer normalization," *arXiv preprint arXiv:1607.06450*, 2016.

[40] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[41] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[42] N. Shazeer and M. Stern, "Adafactor: Adaptive learning rates with sub-linear memory cost," in *International Conference on Machine Learning*. PMLR, 2018, pp. 4596–4604.

[43] K. Bajaj, K. Pattabiraman, and A. Mesbah, "Mining questions asked by web developers," in *Proceedings of the 11th Working Conference on Mining Software Repositories*, 2014, pp. 112–121.

[44] M. J. Islam, G. Nguyen, R. Pan, and H. Rajan, "A comprehensive study on deep learning bug characteristics," in *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2019, pp. 510–520.

[45] Q. Liu, Z. Liu, H. Zhu, H. Fan, B. Du, and Y. Qian, "Generating commit messages from diffs using pointer-generator network," in *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*. IEEE, 2019, pp. 299–309.

[46] D. Gros, H. Sezhiyan, P. Devanbu, and Z. Yu, "Code to comment "translation": Data, metrics, baselining & evaluation," in *2020 35th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2020, pp. 746–757.

[47] S. Chen, X. Xie, B. Yin, Y. Ji, L. Chen, and B. Xu, "Stay professional and efficient: automatically generate titles for your bug reports," in *2020 35th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2020, pp. 385–397.

[48] L. Prechelt, "Early stopping-but when?" in *Neural Networks: Tricks of the trade*. Springer, 1998, pp. 55–69.

[49] S. Banerjee and A. Lavie, "Meteor: An automatic metric for mt evaluation with improved correlation with human judgments," in *Proceedings of the acl workshop on intrinsic and extrinsic evaluation measures for machine translation and/or summarization*, 2005, pp. 65–72.

[50] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu, "Bleu: a method for automatic evaluation of machine translation," in *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*, 2002, pp. 311–318.

[51] J. L. Fleiss, "Measuring nominal scale agreement among many raters." *Psychological bulletin*, vol. 76, no. 5, p. 378, 1971.