Generating Question Titles for Stack Overflow from Mined Code Snippets*

ZHIPENG GAO, Monash University, Australia
XIN XIA, Monash University, Australia
JOHN GRUNDY, Monash University, Australia
DAVID LO, Singapore Management University, Singapore
YUAN-FANG LI, Monash University, Australia

Stack Overflow has been heavily used by software developers as a popular way to seek programming-related information from peers via the internet. The Stack Overflow community recommends users to provide the related code snippet when they are creating a question to help others better understand it and offer their help. Previous studies have shown that a significant number of these questions are of low-quality and not attractive to other potential experts in Stack Overflow. These poorly asked questions are less likely to receive useful answers and hinder the overall knowledge generation and sharing process. Considering one of the reasons for introducing low-quality questions in SO is that many developers may not be able to clarify and summarize the key problems behind their presented code snippets due to their lack of knowledge and terminology related to the problem, and/or their poor writing skills, in this study we propose an approach to assist developers in writing high-quality questions by automatically generating question titles for a code snippet using a deep sequence-to-sequence learning approach. Our approach is fully data-driven and uses an *attention* mechanism to perform better content selection, a *copy* mechanism to handle the rare-words problem and a *coverage* mechanism to eliminate word repetition problem. We evaluate our approach on Stack Overflow datasets over a variety of programming languages (e.g., Python, Java, Javascript, C# and SQL) and our experimental results show that our approach significantly outperforms several state-of-the-art baselines in both automatic and human evaluation. We have released our code and datasets to facilitate other researchers to verify their ideas and inspire the follow up work.

CCS Concepts: • **Software and its engineering → Software evolution**; **Maintaining software**;

Additional Key Words and Phrases: Stack Overflow, Question Generation, Question Quality, Sequence-to-sequence

ACM Reference Format:
Zhipeng GAO, Xin Xia, John Grundy, David Lo, and Yuan-Fang Li. 2019. Generating Question Titles for Stack Overflow from Mined Code Snippets. *ACM Trans. Softw. Eng. Methodol.* 9, 4, Article 39 (March 2019), 37 pages.
https://doi.org/0000001.0000001

*Corresponding Authors: Xin Xia

Authors' addresses: Zhipeng GAO, Monash University, Melbourne, VIC, 3168, Australia, zhipeng.gao@monash.edu; Xin Xia, Monash University, Melbourne, VIC, 3168, Australia, xin.xia@monash.edu; John Grundy, Monash University, Melbourne, VIC, 3168, Australia, john.grundy@monash.edu; David Lo, Singapore Management University, Singapore, Singapore, davidlo@smu.edu.sg; Yuan-Fang Li, Monash University, Melbourne, VIC, 3168, Australia, yuanfang.li@monash.edu.

1049-331X/2019/3-ART39 $15.00
https://doi.org/0000001.0000001

1 INTRODUCTION

In recent years, question and answer (Q&A) platforms have become one of the most important user generated content (UGC) portals. Compared with general Q&A sites such as Quora[1] and Yahoo! Answers[2], Stack Overflow[3] is a vertical domain Q&A site, its content covers the specific domain of computer science and programming. Q&A sites, such as Stack Overflow, are quite open and have little restrictions, which allow their users to post their problems in detail. Most of the questions will be answered by users who are often domain experts.

Stack Overflow (SO) has been used by developers as one of the most common ways to seek coding and related information on the web. Millions of developers now use Stack Overflow to search for high-quality questions to their programming problems, and Stack Overflow has also become a knowledge base for people to learn programming skills by browsing high-quality questions and answers. The success of Stack Overflow and of community-based question and answer sites in general depends heavily on the will of the users to answer others' questions. Intuitively, an effectively written question can increase the chance of getting help. This is beneficial not only for the information seekers, since it increases the likelihood of receiving support, but also for the whole community as well, since it enhances the behavior of effective knowledge sharing. A high-quality question is likely to obtain more attention from potential answerers. On the other hand, low-quality questions may discourage potential helpers [3, 8, 34, 44, 47, 72].

To help users effectively write questions, Stack Overflow has developed a list of quality assurance guidelines[4] for community members. However, despite the detailed guidelines, a significant number of questions submitted to SO are of low-quality [4, 12]. Previous research has provided some insight into the analysis of question quality on Stack Overflow [3, 4, 11, 12, 14, 37, 42, 58, 73, 75]. Correa and Sureka [12] investigated closed questions on SO, which suggest that the good question should contain enough code for others to reproduce the problem. Arora et al. [4] proposed a novel method for improving the question quality prediction accuracy by making use of content extracted from previously asked similar questions in the forum. More recent work [58] studied the way of identifying unclear questions in CQA websites. However, all of the work focuses on predicting the poor quality questions and how to increase the accuracy of the predictions, more in-depth research of dealing with the low-quality questions is still lacking. To the best of our knowledge, this is the first work that investigates the possibility of automatically improving low-quality questions in Stack Overflow. Previous studies [11, 57, 58] have shown that one of the major reasons for the introduction of low-quality questions is that developers do not create *informative* question titles. Considering information seekers may lack the knowledge and terminology related to their questions and/or their writing may be poor, formulating a clear question title and questioning on the key problems could be a non-trivial task for some developers. Lacking important terminology and pool expression may happen even more often when the developer is less experienced or less proficient in English.

Among the Stack Overflow quality assurance guidelines, one of which is that developers should attach code snippets to questions for the sake of clarity and completeness of information, which lead to an impressive number of code snippets together with relevant natural language descriptions accumulated in Stack Overflow over the years. Some prior work has investigated retrieving or generating code snippets based on natural language queries, as well as annotating code snippets using natural language (e.g., [2, 13, 15, 20, 21, 27, 30, 32, 35, 38, 41, 43, 48, 61, 68, 74]). However, to

[1] https://www.quora.com/
[2] https://answers.yahoo.com/
[3] https://stackoverflow.com/
[4] https://stackoverflow.com/help/how-to-ask

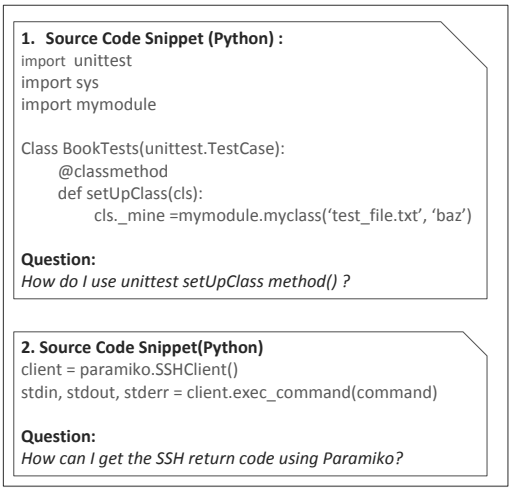

Fig. 1. Example Code Snippet & Question Pairs

the best of our knowledge, there have been no studies dedicated to the question generation[5] task in Stack Overflow, especially generating questions based on a code snippet.

Fig. 1 shows some example code snippets and corresponding question titles in Stack Overflow. Generating such a question title is often a challenging task since the corpus not only includes natural language text, but also complex code text. Moreover, some rare tokens occur among the code snippet, such as "setUpClass" and "Paramiko" illustrated in the aforementioned examples.

We propose an approach to help developers write high-quality questions based on their code snippets by automatically generating question titles from given code snippets. We frame this question generation task in Stack Overflow as a sequence-to-sequence learning problem, which directly maps a code snippet to a question. To solve this novel task, we propose an end-to-end sequence-to-sequence system, enhanced with an *attention* mechanism [5] to perform better content selection, a *copy* mechanism [23] to handle the rare-words problem, as well as a *coverage* mechanism [59] to avoid meaningless repetition. Our system consists of two components: a source-code encoder and a question decoder. Particularly, the code snippet is transformed by a source-code encoder into a vector representation. When it comes to the decoding process, the question decoder reads the code embeddings to generate the target question titles. Moreover, our approach is fully data-driven and does not rely on hand-crafted rules.

To demonstrate the effectiveness of our model, we evaluated it using automatic metrics such as BLEU [49] and ROUGE [40] score, together with a human evaluation for naturalness and relevance of the output. We also performed a practical manual evaluation to measure the effectiveness of our approach for improving the low-quality questions in Stack Overflow. From the automatic evaluation, we found that our approach significantly outperforms a collection of state-of-the-art baselines, including the approach based on information retrieval [52], a statistical machine translation approach [36], and an existing sequence-to-sequence architecture approach in commit message generation [33]. For human evaluation, questions generated by our system are also rated as more natural and relevant to the code snippet compared with the baselines. The practical

[5]"question generation" in this paper is to generate the question titles for a Stack Overflow post.

manual evaluation shows that our approach can improve the low-quality question titles in terms of Clearness, Fitness and Willingness.

In summary, this paper makes the following three main contributions:

- We propose a novel question generation task based on a sequence-to-sequence learning approach, which can help developers to phrase high-quality question titles from given code snippets. Enhanced with the *attention* mechanism, our model can perform the better content selection, with the help of and *copy* mechanism and *coverage* mechanism, our model can manage rare word in the input corpus and avoid the meaningless repetitions. To the best of our knowledge, this is the first work which investigates the possibility of improving the low-quality questions in Stack Overflow.
- We performed comprehensive evaluations on Stack Overflow datasets to demonstrate the effectiveness and superiority of our approach. Our system outperforms strong baselines by a large margin and achieves state of the art performance.
- We collected more than 1M ⟨*code snippet, question*⟩ pairs from Stack Overflow, which covers a variety of programming languages (e.g., Python, Java, Javascript, C# and SQL). We have released our code[6] and datasets [17] to facilitate other researchers to repeat our work and verify their ideas. We also implemented a web service tool, named Code2Que to facilitate developers and inspire the follow-up work.

The rest of the paper is organized as follows. Section 2 presents key related work on question generation and relevant techniques. Section 3 presents the motivation of this study. Section 4 presents the details of our approach for the question generation task in Stack Overflow. Section 5 presents the experimental setup, the baseline methods and the evaluation metrics used in our study. Section 6 presents the detailed research questions and the evaluation results under each research question. Section 7 presents the contribution of the paper and discusses the strength and weakness of this study. Section 8 presents threats to validity of our approach. Section 9 concludes the paper with possible future work.

2 RELATED WORK

Due to the great value of Stack Overflow in helping software developers, there is a growing body of research conducted on Stack Overflow and its data. This section discusses various work in the literature closely related to our work, i.e., deep source code summarization, the empirical study of Stack Overflow on quality assurance, and different tasks by mining the Stack Overflow dataset. It is by no means a complete list of all relevant papers.

2.1 Deep Source Code Summarization

A number of previous works have proposed methods for mining the ⟨natural language, code snippet⟩ pairs, these techniques can be applied to tasks such as code summarization as well as commit message generation. (e.g., [32], [30], [33], [62]).

One similar work with ours is Iyer et at.[32]. They proposed Code-NN, which uses an attentional sequence-to-sequence algorithm to summarize code snippets. This work is similar to our approach because our approach also uses an sequence-to-sequence model. However, there are three key differences between our approach and Code-NN. First, the goal of of Code-NN is summarizing source code snippets while the goal of our approach is generating questions from code snippets. Second, the Code-NN only incorporates attention mechanism while our approach also employs copy mechanism and coverage mechanism, which is more suitable for the specific task of question generation. Third, Code-NN needs to parse the code into AST, while most code snippets in SO are

[6] https://github.com/beyondacm/Code2Que

not parsable (e.g., the example code in Fig. 8). Followed by Iyer's work, Hu et al. [30] proposed to use the neural machine translation model on the code summarization with the assistance of the structural information (i.e., the AST). And Wan et al. [62] applied deep reinforcement learning (i.e., tree structure recurrent neural network) to improve the performance of code summarization. Their approach also use AST as the input. All of the aforementioned studies rely on the AST structure of the source code, and note that most of the code in Stack Overflow are not parsable. Thus, the AST-based approaches can not apply to our work.

2.2 Question Quality Study on Stack Overflow

The general consensus is that the quality of user-generated content is a key factor to attract users to visit knowledge-sharing websites. Many studies have investigated the content quality in Stack Overflow (e.g., [3, 4, 11, 12, 14, 37, 42, 46, 50, 58, 72, 73, 75]).

For example, Nasehi et al. [46] manually performed a qualitative assessment to investigate the important features of precise code examples in answers of 163 SO posts. Yao et at. [73] investigated quality prediction of both Q&As on SO. The output revealed that answer quality is strongly positively associated with that of its question. Yang et al. [72] found that the number of edits on a question is a very good indicator of question quality. Ponzanelli [50] developed an approach to do automatic categorization of questions based on their quality. Correa et al. [11] studied the closed questions in Stack Overflow, finding that the occurrence of code fragments is significant.

All of the above mentioned studies are either predicting quality of the post or increasing the accuracy of predictions. Different from the existing research, our approach is related to improve the quality of the questions. To the best of our knowledge, this is the first work which investigates the possibility of improving the low quality questions using code snippets in Stack Overflow.

2.3 Machine/Deep Learning on Software Engineering

Recently, an interesting direction of software engineering is to use machine/deep learning for different tasks to improve software development. Such as code search (e.g., [2, 24, 31, 39]), clone detection (e.g., [7, 18, 19, 64, 67]), program repair (e.g,. [10, 45, 60, 66]), document (such as API and questions/answers/tags) recommendation (e.g., [22, 25, 26, 55, 63, 65, 69, 70, 76]).

For code search tasks, Gu et al. [24] proposed a deep code search model which uses two deep neural networks to encode source code and natural language description into a vector representation and then uses a cosine similarity function to calculate their similarity. Allamanis et al. [2] proposed a system that uses Stackoverflow data and web search logs to create models for retrieving C# code snippets given natural language questions and vice versa. For clone detection tasks, white et al. [67] first proposed a deep learning-based clone detection method to identify code clones via extracting features from program tokens. For program repair tasks, White et al. [66] propose an automatic program repair approach, DeepRepair, which leverages a deep learning model to identify the similarity between code snippets. For document recommendation tasks, Xia et al. [69] developed a tool, called TagCombine, an automatic tag recommendation method which analyzes objects in software information sites. Gkotsis et al. [22] developed a novel approach to search and suggest the best answers through utilizing textual features. Gangul et al. [16] examined the retrieval of a set of documents, which are closely associated with a newly posted question. Chen et al. [9] studied cross-lingual question retrieval to assist non-native speakers more easily to retrieve relevant questions.

Although the aforementioned studies have utilized machine/deep learning for different software development activities, to our best knowledge, no one has yet considered the question generation task in Stack Overflow. In contrast to all previous work, we propose a novel approach to generate a

question by a given code snippet. Our work is first to tackle such a task for helping developers to generate a question when presenting a given code snippet.

3 MOTIVATION

In this section, we first summarise the problem and our solution in this study. Following that, we present some example user scenarios of employing our approach in the software development process. We then show some motivating examples from Stack Overflow of the sorts of problems our work addresses.

3.1 The Problem and Our Solution

Despite the detailed guidelines provided by the community, a very large number of questions in Stack Overflow are of low-quality [4, 12]. These poorly asked questions are often ambiguous, vague, and/or incomplete, and hardly attract potential experts to provide answers, thus hindering the progress of knowledge generation and sharing. In order to improve question quality, we need to improve title, body and tags. In this work, we focus on improving titles. The motivation for our work is that improving low-quality question titles can potentially be helpful in increasing the likelihood of getting help for the information seekers, as well as reducing the manual effort for quality maintenance of the CQA community. We propose a novel approach to assist developers in posting high-quality questions by generating question titles for a given code snippet. Our approach provides benefit for the following tasks: (i) *Question Improvement*: many developers can not post clear and/or informative questions due to their lack of knowledge and terminology related to the problem, and/or their poor english writing skills. Our approach can generate high-quality question titles for helping developers to summarize the key problems behind their presented code snippet. (ii) *Edits Assistance*: the SO community has employed a collaborative editing mechanism to maintain a satisfactory quality level for the post. However, the editing process may require several interactions between the asker and other community members, thus delaying the answering and even causing questions to sink in the list of open issues. Our approach can be used as an automatic edit assistance tool to improve the question formulation process and reduce the manual effort for quality maintenance. (iii) *Code Embeddings*: Another byproduct of our approach is the code embeddings generated by our approach. In this study, we have collected more than 1M code snippets which covers various programming languages such as Java, Python, Javascript, C#, etc. All the code snippets are embedded into a high-dimensional vector space by our approach. A variety of applications such as code search (e.g., [24, 31, 39]) , summarization (e.g., [30, 32, 33, 62]), retrieval (e.g., [1, 9, 71]), and API recommendation (e.g., [25, 26]) can benefit from the code embeddings used in our study.

3.2 Illustrative User Scenarios

We implement our model as a standalone web application tool, called Code2Que. Developers can copy and paste their code snippet to our tool to generate a question title for the code snippet. Meanwhile, by utilizing the vector representation of the code snippets, Code2Que also retrieves a list of top related questions in Stack Overflow and recommends them to the developers. The usage scenarios of our proposed tool are as follows:

Without Tool. Consider Bob who is a developer, who is learning a new development framework. He is also a non-native English speaker with poor English writing skills. Daily, Bob encounters various programming problems during development. He locates the code that is the root cause of the problem, but he cannot figure it out. Due to his lack of the knowledge and terminology of the development framework being used, he does not even know how to most effectively search for answers to the problem on the Internet. Therefore, he creates a question in Stack Overflow,

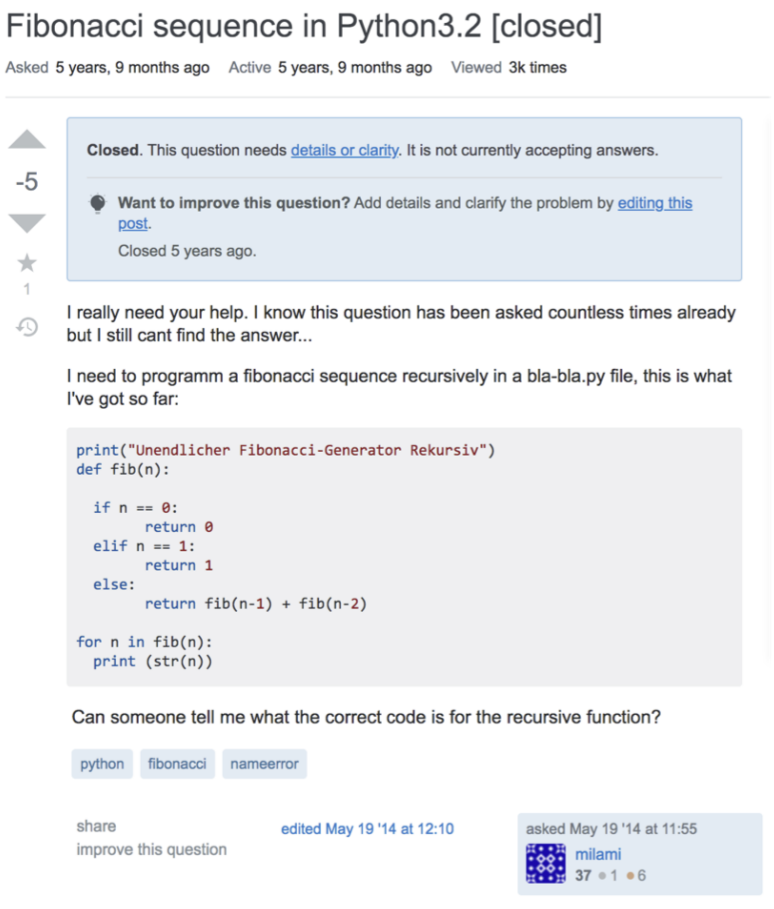

Fig. 2. Example of Problem Questions Title (for Python)

provides his code snippet in the question body according to the Stack Overflow guidelines, and then tries his best to write a question title to summarize the problem. Unfortunately, his question title turns out to be very unclear and uninformative, and there are few users attracted by his question. Bob waits for a long time but does not get any help.

With Tool. Now consider that Bob adopts tool Code2Que. Before he searches on the Internet, Bob copies his code snippet to our Code2Que tool to generate a question title for the code snippet. Bob uses the generated question as a query to search on the internet. The searching results are now closely related to the development framework, even though he is not very familiar with it. Bob can also quickly review a list of related questions in Stack Overflow which have a similar problem code snippet. After going through these results, Bob can gain a better understanding of the problem that he is trying to solve and quickly fix the problem by himself. Moreover, Bob can also go back to his earlier poorly asked questions, Bob can use our tool as an edit assistance tool on question titles for reformulating these low-quality questions. Bob provides the code snippet in the question body and writes a question title based on the question title generated by our tool and the knowledge he learned from the results. This time, his question title is much more clear and informative and Bob's

Fig. 3. Example of Problem Questions Title (for Java)

question soon attracts an expert of the development framework. With the help of this expert, Bob successfully figures his problem out.

3.3 Motivating Examples

A large number of questions have been closed by community members because their question titles are unclear and need further clarifying. For example, the screenshots in Fig. 2 and Fig. 3 show two examples of problematic Stack Overflow question titles. Developers posted a question *"Fibonacci sequence in Python3.2"* and *"I am creating a notepad in java ... to paste it at location of cursor"* in Stack Overflow. They attached their code snippet and tried to explain the key meaning of their problems. However, such question titles are still very uninformative (in Fig. 2) and confusing (in Fig. 3). Both of these questions have been marked as having lack of clarity and need to be further improved upon. Such titles run a real risk of not being found by the ideal people to answer them, may make potential question answering users lose interest, or make users who may answer them have to painstakingly browse the additional paragraph to understand the key point. All reduce the likelihood of them giving help.

Using the tool CODE2QUE described in this paper, we can provide a way to automate the process of improving such poor quality question titles, which is potentially helpful in reducing the manual effort for the quality maintenance of CQA forums. Based on the developer's code snippet, the generated question title by our tool is *"how to find the fibonacci series through recursion?"* for the code snippet shown in Fig. 2 and *"how to change the string value in textarea field using java?"* for the code snippet shown in Fig. 3. These newly generated question titles are much more clear and informative to readers, and also questioning on the key problems of the user's concern. This is helpful for the potential helpers to understand the key problems of the question better and also for the askers to formulate a related question better.

4 APPROACH

In this section, we firstly define the task of question generation, then present the details of Stack Overflow question generation system. Fig. 4 demonstrates the workflow used by our model. A

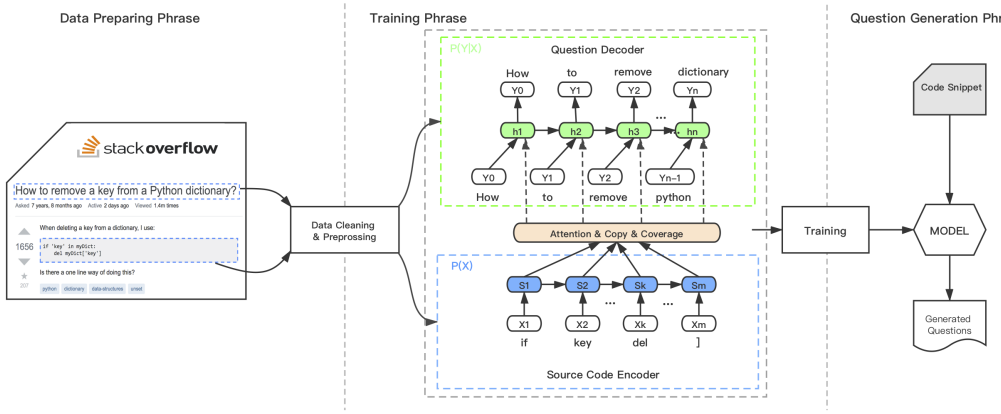

Fig. 4. Workflow of Our Model

Long Short Term Memory (LSTM) encoder-decoder architecture, is enhanced by *attention* mechanism [5], *copy* mechanism [23] and *coverage* mechanism [59]. In general, our model consists of two components: **A Source-code Encoder** and **A Question Decoder**. The source code snippet is transformed by Source-code Encoder into a vector representation, which is then read by a Question Decoder to generate the target question titles. Our model is a differentiable Seq2Seq model with aforementioned three mechanism, i.e., *attention* mechanism, *copy* mechanism and *coverage* mechanism, which can be trained in an end-to-end fashion with gradient descent.

4.1 Question Generation Task Definition

The motivation for our work is to improve the low-quality questions in Stack Overflow. Considering many developers may not be able to describe the problems due to their lack of knowledge and terminology, and/or they are not native english speakers, we propose a novel task in this paper - automatic generation of question titles from a code snippet, the central theme of which is helping developers to create better question titles based on their targets and code snippets. We formulate this task as a sequence-to-sequence learning problem.

Given $\mathbf{C}$ is the sequence of tokens within a code snippet, our target is to generate a Question $\mathbf{Q}$, which is relevant, natural, syntactically and semantically correct. To be more specific, our main objective is to learn the underlying conditional probability distribution $P_\theta(\mathbf{Q}|\mathbf{C})$ parameterized by θ. In other words, the goal is to train a model θ using $\langle$*code snippet, question*$\rangle$ pairs such that the probability $P_\theta(\mathbf{Q}|\mathbf{C})$ is maximized over the given training dataset. More formally given a code snippet $\mathbf{C}$ as a sequence of tokens $(x_1, x_2, ..., x_M)$ of length M, and a question title $\mathbf{Q}$ as a sequence of natural language words $(y_1, y_2, ..., y_N)$ of length N. Mathematically, our task is defined as finding $\overline{y}$, such that:

$$\overline{y} = argmax_{\mathbf{Q}} P_\theta(\mathbf{Q}|\mathbf{C}) \tag{1}$$

where $P_\theta(\mathbf{Q}|\mathbf{C})$ is defined as:

$$P_\theta(\mathbf{Q}|\mathbf{C}) = \prod_{i=1}^{L} P_\theta(y_i|y_1, ..., y_{i-1}; x_1, ..., x_M) \tag{2}$$

$P_\theta(\mathbf{Q}|\mathbf{C})$ can be seen as the conditional log-likelihood of the predicted question title $\mathbf{Q}$ given the input code snippet $\mathbf{C}$.

4.2 Source-code Encoder

Source code token in the code snippet is fed sequentially into the encoder, which generates a sequence of hidden states. Our encoder is a two-layer bidirectional LSTM network,

$$\overrightarrow{\mathbf{f}w_t} = \overrightarrow{\text{LSTM}_2}\left(x_t, \overrightarrow{\mathbf{h}_{t-1}}\right)$$
$$\overleftarrow{\mathbf{b}w_t} = \overleftarrow{\text{LSTM}_2}\left(x_t, \overleftarrow{\mathbf{h}_{t-1}}\right)$$

where x_t is the given input source code token at time step step t, and $\overrightarrow{\mathbf{h}_t}$ and $\overleftarrow{\mathbf{h}_t}$ are the hidden states at time step t for the forward pass and backward pass respectively. The hidden states(from the forward and backward pass) of the last layer of the source-code encoder are concatenated to form a state s as $\mathbf{s} = [\overrightarrow{\mathbf{f}w_t}; \overleftarrow{\mathbf{b}w_t}]$.

4.3 Question Decoder

Our question decoder is a singe-layer LSTM network, initialized with the state s as $\mathbf{s} = [\overrightarrow{\mathbf{f}w_t}; \overleftarrow{\mathbf{b}w_t}]$. Let $qword_t$ be the target word at time stamp t *of the ground truth question title*. During training, at each time step t the decoder takes as input the embedding vector y_{t-1} of the previous word $qword_{t-1}$ and the previous state s_{t-1}, and concatenates them to produce the input of the LSTM network. The output of the LSTM network is regarded as the decoder hidden state s_t, as follows:

$$\mathbf{s}_t = \text{LSTM}_1\left(y_{t-1}, \mathbf{s}_{t-1}\right) \tag{3}$$

The decoder produces one symbol at a time and stops when the END symbol is emitted. The only change with the decoder at testing time is that it uses output from the previous word emitted by the decoder in place of $word_{t-1}$ (since there is no access to a ground truth then).

4.4 Incorporating Attention Mechanism

We model the attention [5] distribution over words in the source code snippets. We calculate the attention (a_i^t) over the i^{th} code snippet token as :

$$e_i^t = v^t \tanh\left(W_{eh}h_i + W_{sh}s_t + b_{att}\right) \tag{4}$$
$$a_i^t = \text{softmax}\left(e_i^t\right) \tag{5}$$

Here, v^t, W_{sh} and b_{att} are model parameters to be learned, and h_i is the concatenation of forward and backward hidden states of source-code encoder. We use this attention a_i^t to generate the context vector c_t^* as the weighted sum of encoder hidden states :

$$\mathbf{c}_t^* = \sum_{i=1,..,|\mathbf{x}|} a_i^t \mathbf{h}_i \tag{6}$$

We further use the c_t^* vector to obtain a probability distribution over the words in the vocabulary as follows,

$$P = \text{softmax}\left(\mathbf{W}_v[s_t, c_t^*] + b_v\right) \tag{7}$$

where W_v and b_v are model parameters. Thus during decoding, the probability of a word is $P(qword)$. During the training process for each word at each timestamp, the loss associated with the generated question title is :

$$Loss = -\frac{1}{T}\sum_{t=0}^{T} logP(qword_t) \tag{8}$$

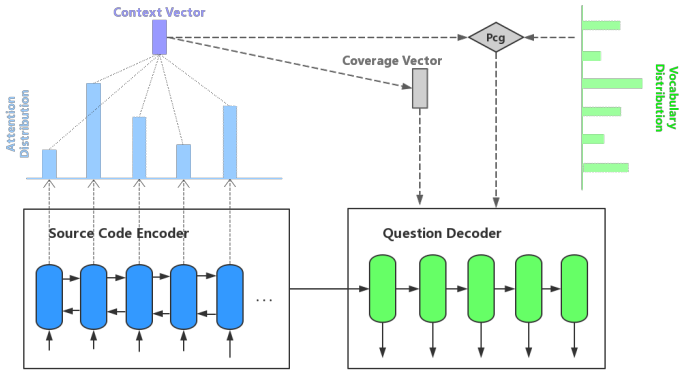

Fig. 5. Attention & Copy & Coverage Mechanism

The *attention* mechanism allows the model to focus on the most relevant parts of the input sequence as needed. For example in Fig. 4, at time step 2, the context vector c_t^* amplifies related hidden states h_k with high scores, and drowning out unrelated hidden states with low scores. For such a case, it enables the question decoder to focus on the word "del" when it generates the word "remove". This ability to amplify the signal from the relevant part of the input sequence makes attention models produce better results than models without attention.

4.5 Incorporating Copy Mechanism

A *copy* mechanism [23] is used to facilitate copying some tokens from the source code snippet to the target generated question title. As illustrated in Fig. 1, some words such as "setUpClass" are naturally going to be much less frequent than other words. Thus it is highly unlikely for a decoder that is solely based on a language model to generate such a word with very rare occurrences in a corpus. In such cases, the possibly rare words in the input sequence might be required to be *copied* from our source code snippet to the target generated question title. We incorporate a *copy* mechanism to handle such rare word problem for Stack Overflow question generation.

In order to learn to copy (from source) as well as to generate words from the vocabulary (using the decoder), we calculate $p_{cg} \in [0, 1]$. This is the decision of a binary classifier that determines whether to generate a word from the vocabulary or to copy the word directly from the input code snippet, based on attention distribution a_i^t:

$$p_{cg} = sigmoid(W_{eh}^T c_t^* + W_{sh}^T s_t + W_x x_t + b_{cg}) \tag{9}$$

Here W_{eh}, W_{sh}, W_x and b_{cg} are trainable model parameters. The final probability of decoding a word is specified by the mixture model :

$$p^*(qword) = p_{cg} \sum_{i:w_i=qword} a_i^t + (1 - p_{cg}) p(qword) \tag{10}$$

where $p*(qword)$ is the final distribution over the union of the vocabulary and the input sequence. As discussed earlier, Equation (10) addresses the rare words issue, since a word not in our vocabulary will have probability $p(qword) = 0$. Therefore, in such cases, our model will replace the $< unk >$ token for out-of-vocabulary words with a word in the input sequence having the highest attention obtained using attention distribution a_i^t. The *copy* mechanism allows the model to locate a certain segment of the input sequence and puts that segment into the output sequence. p_{cg} is a soft switch

to choose between generating a word from vocabulary or copying a word from the input sequence. For example, in Fig. 1, the rare word "setUpClass" in the question title is copied from the input source code snippet. For such a rare word, *copy* mechanism increases the copy-mode probability and decreases the generate-mode probability, which can correctly catch the rare word and put it to the output sequence.

4.6 Incorporating a Coverage Mechanism

Repetition is a common problem for sequence-to-sequence models and to discourage meaningless repetitions, we maintain a word coverage vector *cov*, which is the sum of attention distributions over all previous decoder timesteps:

$$cov^t = \sum_{t'=0}^{t-1} a^{t'} \tag{11}$$

Intuitively, cov^t is a distribution over source code snippet tokens that represents the degree of coverage that those tokens have received from the *attention* mechanism so far. Note that no word is generated before timestamp 0, and hence cov^0 will be a zero vector then. The update equation (4) is now modified to be:

$$e_i^t = v^t \tanh\left(W_{cv} cov_i^t + W_{eh} h_i + W_{sh} s_t + b_{att}\right) \tag{12}$$

Here, W_{cv} are trainable parameters that ensure the *attention* mechanism's current decision is informed by a reminder of its previous decisions. The *coverage* mechanism allows our model to solve the word repetition problem in the output sequence (see Figure 12). The *coverage* mechanism ensures that the *attention* mechanism's current decision is informed by a reminder of its previous decisions (summarized in cov^t). This should make it easier for the *attention* mechanism to avoid repeatedly attending to the same locations, and thus avoid generating repetitive text.

Following the incorporation of the copy and *coverage* mechanism in our attentional sequence-to-sequence architecture, the final loss function will be:

$$Loss = \frac{1}{T} \sum_{t=0}^{T} logP^*(qword_t) + \lambda L_{cov} \tag{13}$$

where λ is a reweighted hyperparameter and the coverage loss L_{cov} is defined as:

$$L_{cov} = \sum_i min(a_i^t, cov_i^t) \tag{14}$$

Once the model is trained, we do inference using a beam search. The beam search is parametrized by the possible paths number k. The inference process stops when the model generates the END token which stands for the end of the sentence.

5 EXPERIMENTAL SETUP

In this section, we firstly describe the evaluation corpus of the task. We then introduce the implementation details of our neural generation approach, the baselines to compare, and their experimental settings. Lastly, we explain the evaluation metrics.

5.1 Pre-processing

We experiment with our neural question generation model on the latest dump of the Stack Overflow (SO) dataset, which is publicly available[7]. Each post comprises a short question title, a detailed question body, and one or more associated answers and multiple tags.

[7] https://archive.org/details/stackexchange

Table 1. Dataset Statistics

Languages	#*Code Tokens*	#*Question Tokens*	Avg.*Code Length*	Avg.*Question Length*
Python	2,367,148	109,329	84.7	11.2
Java	3,371,946	123,994	103.2	10.8
Javascript	2,814,729	121,854	94.1	10.8
C#	2,340,202	100,178	82.1	11.0
SQL	1,483,056	48,668	84.1	10.1

Table 2. Number of Training/Validation/Testing Samples

Python	# pairs (Train)	186,976	# pairs (Test-Raw)	3,000
	# pairs (Val)	3,000	# pairs (Test-Clean)	2,940
Java	# pairs (Train)	250,708	# pairs (Test-Raw)	3,000
	# pairs (Val)	3,000	# pairs (Test-Clean)	2,963
Javascript	# pairs (Train)	290,610	# pairs (Test-Raw)	3,000
	# pairs (Val)	3,000	# pairs (Test-Clean)	2,940
C#	# pairs (Train)	178,830	# pairs (Test-Raw)	3,000
	# pairs (Val)	3,000	# pairs (Test-Clean)	2,974
SQL	# pairs (Train)	150,002	# pairs (Test-Raw)	3,000
	# pairs (Val)	3,000	# pairs (Test-Clean)	2,980

In this study, we performed our experiment on a variety of programming languages, which include Python, Java, Javascript, C# and SQL. To do that, we used the *Python*, *Java*, *Javascript*, *C#* and *SQL* tag for collecting questions associated with the corresponding programming language respectively. Then we removed all questions whose question scores were less than 1. This is reasonable since our goal is to generate high-quality questions to help developers. We extracted code snippets (using ⟨*code*⟩ tags) within the post's question body and corresponding post question title. We added the resulting ⟨*question*, *code snippet*⟩ pairs to our corpus.

5.1.1 Data Preprocessing. We tokenized the code snippet with respect to each programming language for pre-processing respectively. We adopted the NLTK toolkit [6] to separate tokens and symbols. One of the challenging tasks during the tokenization was the structural complexity of the code snippet in our dataset. We stripped out all comments by using the regular expression for different programming languages. After that, in order to avoid being context-specific, numbers and strings within a code snippet and replaced them with special tokens "VAR", "NUMBER" and "STRING" respectively. Table 1, Fig. 6 and Fig. 7 shows some data statistics on the processed dataset. We can see that the length of Java and Javascript code snippets are much longer than the other programming languages. On average, Java and Javascript code snippets contain 103 and 94 tokens respectively, while the code snippets of the other three programming languages are just around 84 tokens long. On the other hand, the question titles of all the programming languages are approximately at the same level, the overall average of the question titles are 11 tokens long.

5.1.2 Data Filtering. Users can post different types of questions in SO, such as "how to X" and "What/Why is Y". In our preliminary study, we targeted questions which include interrogative keywords such as "how", "what", "why", "which", "when". For the above collection of question-code pairs, only the pairs where the aforementioned keywords appear in the question title were kept. After that, we removed pairs where the code snippets are too long or too short. Based on the

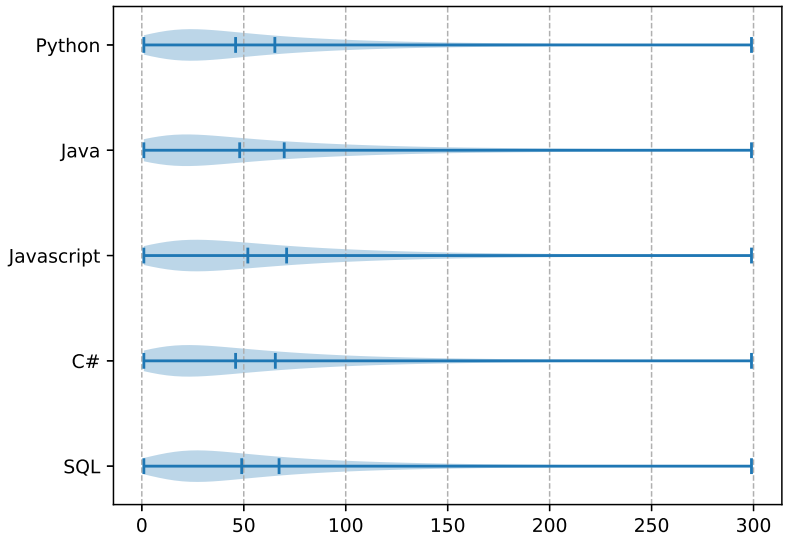

Fig. 6. Volinplots of Code Distribution

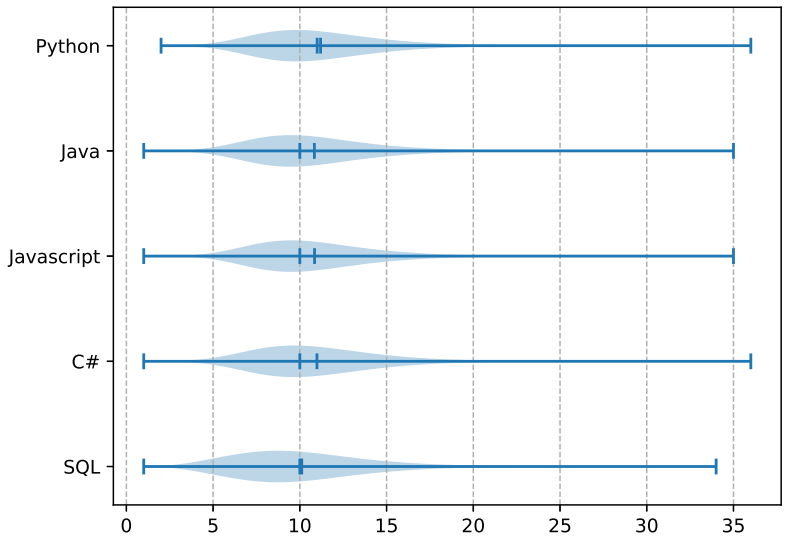

Fig. 7. Volinplots of Question Distribution

interquartile range (IQR) of the violin plots in Fig. 6 and Fig. 7, we only preserved pairs where the token range from 16 tokens to 128 tokens for code snippet and the token range from 4 tokens to 16 tokens for question titles. At this stage, we collected more than 1M ⟨*question*, *code snippet*⟩ pairs in total for Python, Java, Javascript, C# and SQL programming languages. We randomly sampled 3,000 pairs for validation and 3,000 pairs for testing respectively, and kept the rest for training. The details of the training, validation and testing samples for each programming language are summarized in Table 2.

Table 3. Clone Detection Analysis

Similarity	Python	Java	Javascript	C#	SQL
$s_i \in [0.0, 0.2)$	2,153	2,241	1,939	2,328	2,359
$s_i \in [0.2, 0.4)$	512	473	651	422	413
$s_i \in [0.4, 0.6)$	195	187	272	182	159
$s_i \in [0.6, 0.8)$	80	62	78	42	49
$s_i \in [0.8, 1.0]$	60	37	60	26	20

5.1.3 Clone Detection. Considering that there may be duplicate and/or very similar ⟨*code snippet*, *question*⟩ pairs between the training set and testing set, this may mislead the evaluation results. We further conducted a primitive clone detection analysis to remove the noisy examples from our testing data set. A lot of methods have been proposed for clone detection in recent years (e.g., [7, 18, 19, 64, 67]). We followed the approach proposed by [18] for clone detection. For each code snippet, we compose a numerical vector by summing up the word embedding vectors for all the relevant tokens within the code snippet. Then the similarity between two code snippets C_1 and C_2 can be calculated as follows:

$$Distance(C_1, C_2) = Euclidean(e_1, e_2) \tag{15}$$

$$Similarity(C_1, C_2) = 1 - Distance(C_1, C_2) \tag{16}$$

where e_1 and e_2 are the corresponding code embedding vectors of C_1 and C_2. Each code snippet C_i in the testing set is queried against all the code snippets in the training set, the maximum similarity score s_i associated with the C_i is retrieved. The results of s_i with respect to each programming language are summarized in Table 3. If the similarity score s_i is over a threshold δ (δ is set to 0.8 in this study), then the code snippet C_i is viewed as a code clone and will be deleted from our testing set. From the table we can see that the number of clone code snippets is very small, while most code snippets get relatively low similarity scores. After removing all the examples with similarity scores above 0.8 from the testing set, we reconstructed a clean testing set for each programming language, the final results are summarized in Table 2. The clean testing set is used for the final evaluation of this study.

5.2 Implementation Details

We implemented our system in Python using Tensorflow framework. We added special START and END tokens for each sequence in our training set. The vocabulary size for the Java and Python dataset were set to 50,000 and 80,000 respectively. We use a two-layer bidirectional LSTM for the encoder and a single-layer LSTM for the decoder. We set the number of LSTM hidden states to 256 in both encoder and decoder. We choose the word embeddings of 300 dimensions. Optimization is performed using stochastic gradient descent (SGD) with a learning rate of 0.01. We fix the batch size for updating to be 32. During decoding, we perform beam search with beam size of 10. We train the model for 30 epochs. Our hyper-parameters were tuned on the validation set, the evaluation results were reported on the test set. We discuss the details of the parameter tuning in Section 6.

5.3 Baselines

To demonstrate the effectiveness of our proposed approach, we compared it with several competitive baseline approaches. We adapted these approaches slightly for our problems, i.e., generating question titles from a given code snippet. We briefly introduced these approaches and the experimental

settings as below. For each method mentioned below, the involved parameters were carefully tuned, and the parameters with the best performance were used to report the final comparison results.

(1) **IR** stands for the information retrieval baseline. For a given code snippet c_i, it retrieves the *question titles* associated with the code c_j that is closest to the input code c_i from the training set. We use TF-IDF [52] metric to calculate the distance between two code snippets, and build a nearest neighbor model to retrieve the most similar instance from the traning set.
(2) **MOSES** [36] is a widely used phrase-based statistical machine translation system. Here, we treat a tokenzied code snippet as the source language text, and the corresponding question title as the target language text. We run the translation from code snippets to question titles. We train a 3-gram language model on target side texts using KenLM [28], and perform turning with MERT on dev set.
(3) **NMT** Jiang et al. [33] proposed an sequence-to-sequence approach to generate commit message from code, we refer to it as NMT in our study. We choose NMT as one comparing approach since its promising performance in commit generation. NMT model take source code as inputs and associated question title as outputs. Hyperparameters are tuned with validation set.
(4) **CODE-NN** Iyer et al. [32] proposed an attention-based Long Short Term Memory (LSTM) neural network, named CODE-NN, to generate descriptive summaries for C# code snippets and SQL queries. In order to use CODE-NN, the C# code fragments and SQL statements first need to be parsed by the modified version of parser. Considering code snippets in SO are usually incomplete and not parsable, and it is non-trivial to design specific parser to parse code snippets of various programming languages, we tried our best to apply our approach to the CODE-NN dataset, which include 60k+ C# (title, query) pairs and 30k+ SQL (title, query) pairs respectively.

5.4 Evaluation Metrics

We evaluate our task with automatic evaluation, and also perform human evaluation via a user study.

(1) **Automatic Evaluation** To evaluate different models, We adopt BLEU-1, BLEU-2, BLEU-3, BLEU-4 [49], ROUGE-1, ROUGE-2 and ROUGE-L [40] scores. BLEU is a precision-oriented measure commonly used in translation tasks, which measures the average n-gram precision on a set of reference sentences, with a penalty for overly short sentences. BLEU-n is the BLEU score that uses up to n-grams for counting co-occurrences. ROUGE is a recall-oriented measure widely used in summarization tasks, which used to evaluate n-grams recall of the summaries with gold-standard sentences as references. ROUGE-1 and ROUGE-2 measures the uigram and bigrams between the system and reference summaries. ROUGE-L is a longest common subsequence measure metric, it does not require consecutive matches but in-sequence matches that reflect sentence level word order. We conducted a large scale automatic evaluation over various kinds of programming languages, i.e., Python, Java, Javascript, C# and SQL. In our work, we regard the generated *question titles* as candidates, and the original human written question titles as gold-standard references.
(2) **Human Evaluation**Since automatic evaluation of generated text does not always agree with the actual human-perceived quality and usefulness of the results, we also perform human evaluation studies to measure how humans perceive the generated questions. To do this, we consider two modalities in our user study : *Naturalness* and *Relevance*. *Naturalness* measures the grammatical correctness and fluency of the question title generated. *Relevance* measures how relevant the *question title* is to the code snippet, and indicates the factual divergence

of the code snippet to the reference question titles. We randomly sampled 50 ⟨*code snippet*, *question*⟩ pairs from Python and Java test results respectively, for each code snippet, we provided 5 associated *question titles*: one was generated by human (the ground truth *question title*), while the others were generated by baseline methods and our approach. Then we invited 5 evaluators, including 4 Ph.D students and 1 Masters student, all of whom are not co-authors, majoring in Computer Science and have industrial experience with Python as well as Java programming (ranging from 1-3 years). All of the five evaluators have at least one year of studying/working-experience in English speaking countries. Each participant was asked to manually rate generated question titles on a scale between 1 and 5 (5 for the best results) across the above modalities. The volunteers were blinded as to which question title was generated by our approach.

(3) **Practical Manual Evaluation** Following the human evaluation, we also performed a practical manual evaluation to further analyze whether our approach can generate better question titles for *low-quality* questions in Stack Overflow. To do this, we randomly sampled 50 *low-quality* ⟨*code snippet*, *question*⟩ pairs from our Python and Java datasets before the data preprocessing. It is worth mentioning that different from human evaluation, these sampled posts were not included in our training and/or testing set, because all the questions with score less than 1 were removed before training processing. For each code snippet, we applied our approach to generate a question title for manual annotation. We conducted pairwise comparison between two question titles (one was generated by humans, one was generated by our tool) for the same code snippet. For each pairwise comparison, we asked the same 5 evaluators to decide which one is better or non-distinguishable in terms of the following three metrics: *Clearness*, *Fitness*, *Willingness to Respond*. *Clearness* measures whether a question title is expressed in a clear way. Unclear questions are ambiguous, vague, and/or incomplete. *Fitness* measures whether a question title is reasonable in logic with the provided code snippet, and whether it is questioning on the key information. Unfit question titles are either irrelevant to the code snippet or universal questions. *Willingness to Respond* measures whether a user is willing to respond to a specific question. This metric is used to justify how likely the generated questions can elicit further interactions. If people are willing to respond, the interactions can go further. Each metric is evaluated independently on each pairwise comparison. Also the two question titles were randomly shuffled and the participants do not know which question is generated by our approach.

6 RESULTS AND ANALYSIS

To gain a deeper understanding of the performance of our approach, we conduct analysis on our evaluation results in this section. For quantitative analysis, firstly we study the experimental results of automatic evaluation, then we examine the outcome of human evaluation. Specifically, we mainly focus on the following research questions:

- *RQ-1:* How effective is our approach under automatic evaluation?
- *RQ-2:* How effective is our approach compared with the CODE-NN model?
- *RQ-3:* How effective is our approach under human evaluation?
- *RQ-4:* How effective is our approach for improving low-quality questions?
- *RQ-5:* How effective is our use of *attention* mechanism, *copy* mechanism and *coverage* mechanism under automatic evaluation?
- *RQ-6:* How effective is our approach under different parameter settings?
- *RQ-7:* How efficient is our approach in practical usage?

Table 4. Automatic evaluation(Python dataset)

Model	BLEU-1	BLEU-2	BLEU-3	BLEU-4	ROUGE-1	ROUGE-2	ROUGE-L
IR_{TFIDF}	20.2 ± 1.1%	17.7 ± 0.4%	18.4 ± 0.3%	18.0 ± 0.2%	24.4 ± 1.4%	6.9 ± 0.6%	21.8 ± 1.2%
Moses	20.4 ± 1.4%	18.1 ± 0.8%	17.8 ± 0.7%	17.4 ± 0.6%	26.9 ± 1.3%	6.2 ± 0.5%	20.4 ± 1.1%
NMT	28.9 ± 1.7%	21.9 ± 0.7%	21.3 ± 0.3%	20.3 ± 0.2%	34.1 ± 2.2%	10.6 ± 1.1%	31.2 ± 1.9%
Ours	**35.8 ± 2.0%**	**30.1 ± 0.9%**	**26.8 ± 0.4%**	**24.2 ± 0.3%**	**39.9 ± 2.5%**	**12.6 ± 2.5%**	**36.7 ± 2.4%**

Table 5. Automatic evaluation(Java dataset)

Model	BLEU-1	BLEU-2	BLEU-3	BLEU-4	ROUGE-1	ROUGE-2	ROUGE-L
IR_{TFIDF}	18.1 ± 1.1%	17.2 ± 0.5%	18.0 ± 0.4%	17.6 ± 0.3%	22.2 ± 1.3%	6.2 ± 0.7%	19.9 ± 1.2%
Moses	18.5 ± 1.0%	17.3 ± 0.6%	17.1 ± 0.5%	16.7 ± 0.4%	25.2 ± 1.5%	5.3 ± 0.4%	20.6 ± 1.2%
NMT	25.0 ± 1.6%	20.7 ± 0.7%	20.9 ± 0.3%	20.2 ± 0.2%	30.0 ± 2.0%	9.6 ± 1.1%	27.3 ± 1.8%
Ours	**31.8 ± 1.8%**	**27.5 ± 0.7%**	**25.2 ± 0.3%**	**23.3 ± 0.2%**	**35.4 ± 2.2%**	**10.0 ± 1.8%**	**32.6 ± 2.1%**

Table 6. Automatic evaluation(Javascript dataset)

Model	BLEU-1	BLEU-2	BLEU-3	BLEU-4	ROUGE-1	ROUGE-2	ROUGE-L
IR_{TFIDF}	18.7 ± 1.1%	17.6 ± 0.4%	18.3 ± 0.3%	17.9 ± 0.2%	22.6 ± 1.3%	6.2 ± 0.6%	20.2 ± 1.1%
Moses	18.9 ± 1.2%	18.8 ± 0.7%	18.7 ± 0.7%	18.3 ± 0.6%	25.7 ± 1.2%	5.8 ± 0.4%	20.1 ± 1.0%
NMT	28.1 ± 1.6%	22.0 ± 0.6%	21.5 ± 0.3%	20.5 ± 0.2%	32.8 ± 1.9%	10.3 ± 1.0%	30.4 ± 1.7%
Ours	**33.2 ± 1.9%**	**26.4 ± 0.8%**	**24.1 ± 0.4%**	**22.1 ± 0.3%**	**37.3 ± 2.2%**	**11.7 ± 1.8%**	**34.7 ± 2.1%**

Table 7. Automatic evaluation(C# dataset)

Model	BLEU-1	BLEU-2	BLEU-3	BLEU-4	ROUGE-1	ROUGE-2	ROUGE-L
IR_{TFIDF}	18.0 ± 1.0%	17.1 ± 0.4%	17.9 ± 0.3%	17.6 ± 0.2%	21.9 ± 1.3%	6.3 ± 0.6%	19.9 ± 1.1%
Moses	18.5 ± 1.0%	16.8 ± 0.7%	16.6 ± 0.6%	16.3 ± 0.6%	25.4 ± 1.2%	6.0 ± 0.4%	20.0 ± 1.0%
NMT	24.4 ± 1.7%	19.3 ± 0.7%	19.8 ± 0.2%	19.3 ± 0.2%	29.4 ± 1.6%	9.7 ± 0.8%	27.1 ± 1.4%
Ours	**30.9 ± 1.8%**	**27.7 ± 0.7%**	**25.3 ± 0.3%**	**23.4 ± 0.2%**	**34.8 ± 2.3%**	**10.2 ± 1.9%**	**31.8 ± 2.2%**

6.1 RQ-1: How effective is our approach under automatic evaluation?

6.1.1 Automatic Evaluation Results. The automatic evaluation results of our proposed model and aforementioned baselines are summarized in Table 4, 5, 6, 7, 8 for Python, Java, Javascript, C#, and SQL respectively. The best performing system for each column is highlighted in boldface. As can be seen, **our model outperforms all the other methods considerably** in terms of BLEU score and ROUGE score. BLEU score measures precision of the system. To be more specific, it measures how many words (and/or n-grams) in the machine generated question titles appear in the ground truth question titles. For ROUGE scores, it measures the recall of the system i.e. how many words(and/or n-grams) in the ground-truth question titles appear in the machine generated questions titles. From the table, we can observe the following points:

(1) In general, encoder-decoder architecture baselines, i.e., NMT and our proposed methods, outperform both the IR based approach and the statistical machine translation approach (e.g., Moses) by a large margin. For IR based approach, it retrieves questions from existing database according to similarity score, which relies heavily on whether similar code snippets can be found and how similar the code snippets are. As a result, it is unable to consider the context of the code snippet, which is reflecting that memorizing the training set is not enough for this task. For the phrase-based statistical approaches which use separately engineered subcomponents, the encoder-decoder model uses the vector representation for words and internal states, semantic and structural information can be learned from these vectors by taking global context into consideration.

Table 8. Automatic evaluation(SQL dataset)

Model	BLEU-1	BLEU-2	BLEU-3	BLEU-4	ROUGE-1	ROUGE-2	ROUGE-L
IR_{TFIDF}	15.6 ± 1.0%	17.6 ± 0.4%	18.4 ± 0.3%	17.9 ± 0.3%	19.3 ± 1.2%	3.7 ± 0.6%	16.4 ± 1.0%
Moses	17.3 ± 0.9%	16.6 ± 0.7%	16.5 ± 0.6%	16.2 ± 0.6%	21.4 ± 1.1%	3.4 ± 0.3%	15.0 ± 0.8%
NMT	22.0 ± 1.3%	20.4 ± 0.5%	20.7 ± 0.4%	19.9 ± 0.2%	26.6 ± 1.7%	7.4 ± 1.0%	22.9 ± 1.5%
Ours	**26.8 ± 1.6%**	**23.8 ± 0.6%**	**22.6 ± 0.3%**	**21.2 ± 0.2%**	**30.5 ± 2.0%**	**8.4 ± 1.3%**	**26.3 ± 1.9%**

(2) Regarding the BLEU score, our approach is significantly better than the other methods (e.g., traditional IR method, phrase-based statistical method, and NMT methods) and achieves understandable results [54]. For example, it improves over NMT methods on BLEU-4 by **19.2%**[8] on Python dataset and **15.3%** on Java dataset. We attribute this to the following reasons: firstly, our approach is based on a sequence-to-sequence architecture and hence it is superior to the statistical baselines[36]. Secondly, compared with NMT baseline which is solely based on the sequence-to-sequence approach, besides using the encoder-decoder architecture, our approach also incorporates an *attention* mechanism to perform better content selection, a *copy* mechanism to manage the rare-words problem in source code snippet, as well as a *coverage* mechanism to eliminate meaningless repetitions, which makes it superior to the NMT baselines. According to [54], the bleu-1 score above 0.30 generally reflect understandable results and above 0.50 reflect good and fluent translations, the bleu score of our approach can be considered as acceptable, but there is still a large gap compared with ground truth question titles.

(3) Regarding the ROUGE score, the advantage of our proposed model is also clear. The potential explanation is that baseline methods, such as Moses, NMT, even with a much larger vocabulary, still has a large number of out of vocabulary words. Our model, augmented with the *copy* mechanism to handle the rare-words problem, beats these baselines by a large margin. This further justifies that the *copy* mechanism generally helps when dealing with the question generation tasks. It also signals that out of vocabulary tokens within code snippet convey much valuable information when generating question titles.

(4) The proposed approach performs best on the Python dataset and worst on the SQL dataset. This is in part because, compared with Python code snippet, SQL code snippets only contain a set of keywords and functions, and thus generating question titles for SQL code snippet is more challenging for solely relying on the compositional structures in the input.

6.1.2 Examples of the Automatic Evaluation. We examine several sample outputs by hand to perform a further qualitative analysis. Fig. 8 shows some examples of the question titles generated by human (Golden questions), the baselines (e.g., IR, Moses and NMT) and our approach for the given code snippets in the test set. We have the following interesting observations:

(1) We see a large gap between our approach and other baselines. **Our approach generates syntactically and semantically correct and relevant question titles in most cases**, while the outputs of every other model are less meaningful and/or more irrelevant. This is consistent with our previous automatic evaluation results. For the IR method, often the question titles are unable to connect to the code snippet. For example in the third sample, the ground truth question is about "find difference between two values", while the IR methods retrieved the question of "how to calculate the diff between two dates in django". The statistical machine translation model, such as Moses, is unable to generate a syntactically correct question title. For example, in the sixth and seventh sample, the question titles

[8]The improvement ratio is defined within https://www.d.umn.edu/~gshute/arch/improvements.xhtml

ID	Code Snippet	Questions
1	import unittest import sys Class BookTests(unittest TestCase): @classmethod def setUpClass(cls): cls._mine = mymodule.myclass('test_file.txt', 'baz')	**Golden** : How do I use unittest setUpClass method() ?
		IR :how to run code when a class is subclassed?
		Moses : how do i import a class booktests unit test how to mine and python
		NMT : how do i write a pytest from testsuit ?
		Ours : how do i use a unittest setup class method() ?
2	import win32gui ImportError : No module named win32gui	**Golden** : How to use the win32gui module with Python ?
		IR : import error: no module named numpy
		Moses : python win com guiimport guiwin gui importerror module in windows ?
		NMT : how to choose window to a python script to window with window ?
		Ours : how to install win32gui . in windows without

3	def distance(x, y):
	if x >= y:
	result = x - y
	else:
	result = y - x
	return result
		Golden : How do I find the difference between two values without knowing which is larger?

		IR : How to calculate diff between two dates in django
		Moses : how do I get return to print to console in this code ?
		NMT : how to make a python program that is not a list of list ?
		Ours : How to find the absolute distance of two point in python ?
4	In[2] : mimetypes. guess_extension('image/jpeg', strict=False) Out[2] : '.jpe'	**Golden** : Why the various JPEG Extensions?
		IR : How to load JPE image file?
		Moses : how to extension in how to include header in python
		NMT : how to get the index of a list of a list of list in python ?
		Ours : how to safely get the file extensions from a file use django ?
5	<?xml version="1.0" encoding="UTF-8"?> <MyDocument xmi:version="2.0"> <Thingamabob name="A" hasDohicky="//@Dohicky.0"> <Dingus/> </Thingamabob> <Dohicky name="B"/> </MyDocument>	**Golden** : In XML what do you call this: //@Dohicky.0 and how to address it in Java
		IR : Why should you use XML CDATA blocks?
		Moses : how to use uniqueconstraint with single table inherite jpa ?
		NMT : how to get the current time in java ? is not abl to get the ip address and host name ?
		Ours : how to get the attribute of node and its value in xml use dom in java ?

6	String path = "/puppy.png"
	try {
	BufferedImage image = ImageIO.read(
	getClass().getResourceAsStream(path));
	} catch (Exception ex) {
	ex.printStackTrace();
	}
		Golden : java input == null why ?

		IR : how do I generate random integers within a specific range in java?
		Moses : file pixel in java ?
		NMT : how to read a file from a file in java ? is not abl to do so
		Ours : how to get the path of an image in java ?
7	webbrowser.open('STRING') gmail_user = raw_input('Please enter your Gmail username:')	**Golden** : How can I disable webbrowser message in python ?
		IR : how to throw custom 404 messages in python
		Moses : how to input
		NMT : how to open a file from a file use python ?
		Ours : how to I open the web browser message when python2 ?

8	def test1():
	exec('print "hi from test1"')
	
	def test2():
	exec('print "hi from test2"')
	def subfunction():
	return True
		Golden : Why doesn'tt exec work in a function with a subfunction?

		IR : Why does Python code run faster in a function?
		Moses : how to test work in a function with a subfunction ? python
		NMT: in python, why does't the alternative of a function with a subfunction ?
		Ours : how python, what is this, ? this function ? some subfunction ?

Fig. 8. Examples of output generated by each model

generated by Moses are incomplete and meaningless. For the NMT method, although it can generate the question titles in the right format in some cases, it still fails to replicate the critical tokens (e.g., example1) because of the difficulty brought by the unseen words in the code snippet.

(2) **Our approach handles out of vocabulary words well**, and it can generate acceptable question titles for a code snippet with rare words. In contrast, the baseline methods often fail in such cases. For example, in the first sample, in which the focus should be put on "setUpClass" method in the code snippet, Our model successfully captures this rare phrase, while other baselines return non-relevant descriptions. It is quite interesting that our model automatically

learns to select informative tokens in the code snippet, which shows the extractive ability of our model. At the same time, our approach often generates words to "connect" those critical tokens, showing its aspect of abstractive ability.

(3) **A large number of the question titles generated by our model produce meaningful output for simple code snippets**. Note that in some cases, the generated question titles are not exactly inline with the standard ones, yet still make sense by looking at the meaning of the code snippet. For example, in the second case, the ground truth question title is "How to use win32gui module with Python", our system generates a question title about "how to install win32gui". This is reasonable given the source code contains "ImportError" while "import win32gui". In the third case, our approach generates a question title of "how to find the absolute distance of two point in python", this is because the code snippet defines a function that returns the distance of two points. For such cases, it is reasonable to generate different question titles that look at the code snippet from different aspects. Our question titles can also be viewed as correct and meaningful by looking at the meanings of the code snippet.

(4) Sometimes, **our approach can generate *question titles* that are more clear and informative than the ground truth question titles**, such as samples 4-6. For example, in the fourth sample, the ground truth question title is "why the various JPEG extensions?" which is uninformative and unclear to the potential helpers, after using our tool the question title can be rephrased as "how to safely get the file extensions from a file" which is more attractive and informative than the original ones.

(5) However, **outputs from our system are not always "correct"**. For example, in the last second sample, the ground truth question title is "How can I disable the web browser message in python", however, our system output an "opposite" *question title* of "How to I open the web browser message when python2". This example reveals that in some cases, question titles can be generated incorrectly by only looking at the implementation details of the code snippet. This is because we can not judge the developers' intent just through the code snippet attached to the question.

(6) Also, **outputs from our system are not always "perfect"**. The gap between ground truth question titles and machine generated question titles is still large. For example, in the last sample, The question quality of our model degrades on longer and compositional inputs. This indicates that there is still a large room for our question generation system to improve. It would be interesting to further investigate how to interpret why certain irrelevant words are generated in the question title. For example, in the second and fifth samples, there are some irrelevant words at the end of generated questions. We will address such problems in the future.

Answer to *RQ-1:* How effective is our approach under automatic evaluation? - we conclude that our approach is effective under automatic evaluation and beats the baselines by a large margin.

6.2 RQ-2: How effective is our approach compared with the CODE-NN model?

CODE-NN trained a neural attention model generate summaries of C# and SQL code fragment, they have published their C# and SQL datasets, which include 66,015 (title, query) pairs for C# and 32,337 pairs for SQL. It is worth emphasizing that CODE-NN removed all the non-parsable code snippets and retained only the parsable code snippets. We retrained our approach on the CODE-NN datasets, the automatic evaluation results of our approach and CODE-NN model are summarized in Table 9. Because CODE-NN use the BLEU-4 metric for evaluation, we only report the BLEU-4 score

Table 9. Automatic evaluation(CODE-NN dataset)

Model	BLEU-4 (C# Dataset)	BLEU-4 (SQL Dataset)
IR	13.7	13.5
Moses	11.6	15.4
CODE-NN	20.5	18.4
Ours	22.1	20.4
Ours (Transfer)	21.3	18.4

in our table. Apart from that, we also explored the effectiveness of transferring our trained model to the new datasets. We further applied the C# and SQL model already obtained to the CODE-NN datasets. This is reasonable because CODE-NN extracted the code snippet only from the accepted answers containing exactly one code snippet, while our approach extracted the code snippet from the questions, so training dataset of our approach will not contaminate the CODE-NN datasets. In other words, our model does not see any test case in the CODE-NN dataset during the training process. From the table, we can observe the following points:

(1) In general, our approach and CODE-NN outperforms the other baselines by a large margin. The results are consistent with our previous evaluation. This further justifies the encoder-decoder architecture approach is helpful to learn the semantic and structural information from the code snippet.
(2) The neural models, i.e., CODE-NN and ours, have better performance on C# than SQL. This is probably due to the following reasons: First, generating question titles for SQL code snippets is a more challenging task since the SQL code snippet only has a handful of keywords and functions, and the generation models need to rely on other structural aspects. Second, the size of the SQL training data (32,337 pairs) is much smaller than the size of the C# training data (66,015 pairs), it is more difficult to train a good neural model if there is lack of sufficient training data.
(3) By using CODE-NN datasets, our model performs better than CODE-NN. It improves BLEU-4 score by 7.8% on C# dataset and 10.8% on SQL dataset. We attribute this to the *copy* mechanism and *coverage* mechanism incorporated into our approach, which is able to handle the low frequency tokens and reduce the redundancy during the generation process.
(4) By transferring existing trained models to the CODE-NN datasets, it is notable that even without training directly on the CODE-NN datasets, we can still achieve comparable results compared with the CODE-NN model. We attribute this to the advantage of our model as well as the larger datasets constructed with our approach. We have collected more than 170K ⟨*code snippet*, *question*⟩ pairs for C# and more than 150K pairs for SQL. The CODE-NN datasets only include 60k+ C# pairs and 30k+ SQL pairs. This verifies the importance of using big training data for applying deep learning-based methods in software engineering.

Answer to *RQ-2:* How effective is our approach compared with CODE-NN? - we conclude that our approach is more effective compared with Code-NN.

6.3 RQ-3: How effective is our approach under human evaluation?

6.3.1 Human Evaluation Results. Fig. 9 shows one example in our human evaluation study. We obtain 250 groups of scores from human evaluation for Python and Java Dataset respectively. Each group contains 4 pairs of scores, which were rated for candidates produced by IR, Moses, Seq2Seq and our approach. Each pair contains a score for the *Naturalness* modality and a score for *Relevance* modality. We regard a score of 1 and 2 as low-quality, a score of 3 as medium quality, and a score of 4

```
    DefaultHttpClient httpclient = new DefaultHttpClient();
CookieStore cookieStore = httpclient.getCookieStore();
BasicClientCookie cookie = new BasicClientCookie("abc", "123");

// Prepare a request object
HttpGet httpget = new HttpGet("http://abc.net/restofurl");

cookieStore.addCookie(cookie);
httpclient.setCookieStore(cookieStore);

// Execute the request
HttpResponse response = httpclient.execute(httpget);

// Examine the response status
log.info("Http request response is: " + response.getStatusLine());

List<Cookie> cookies = cookieStore.getCookies();

for (int i=0; i<cookies.size();i++) {

    if (cookies.get(i).getName().toString().equals("abc")) {
        log.info("cookie is: " + cookies.get(0).getValue().toString());
        }
}
```

Please rate each Candidate for N(Naturalness) and R(Relevance) from 1-5 (5 is the best)		
Reference : Using apache httpclient how to set cookie for http request ?		
Candidate1 : how to connect android app with mysql database through php	N:	R:
Candidate2 : how to use the java httpclient . x how to imit send from us ?	N:	R:
Candidate3 : how to get the url from a http post request ? is not work	N:	R:
Candidate4 : how to get cookie from apache httpclient ?	N:	R:

Fig. 9. User Study Case (Human Evaluation)

and 5 as high-quality. Regarding human evaluation study results, the responses from all evaluators is then averaged for each modality. We also count the proportion of each quality type within each modality. The quality distribution and average score of Naturalness and Relevance across each methods are presented in Table 10 and Table 11. From the table, several points stand out:

(1) From Naturalness prospective, **IR performs a slightly better than our approach**. This is reasonable since it retrieves other similar question titles which are all also written by humans. However its output lacks the explanation to the actual input code snippet, which also explains its surprisingly low score on Relevance.
(2) From Relevance prospective, **the question titles generated by our approach are much more appreciated** by the volunteers. Its superior performance in terms of Relevance further supports our claim that it manages to select content from input more effectively.
(3) In general, **our model performs well across both dimensions**. The results of human evaluation are consistent with automatic evaluation results. The considerable proportion of high-quality questions generated by our approach with respect to the *Naturalness* and *Relevance* also reconfirms the effectiveness of our system.

Answer to *RQ-3:* How effective is our approach under human evaluation? In general, for considering the combination of both modality, i.e., *Naturalness* and *Relevance*, our model beats the baselines by a large margin.

Table 10. Human Evaluation(Python dataset)

Model	Naturalness	Low_N	$Medium_N$	$High_N$	Relevance	Low_R	$Medium_R$	$High_R$
IR	3.91	13.2%	15.6%	71.2%	2.22	66.4%	19.2%	14.4%
Moses	2.44	62.4%	17.2%	20.4%	2.73	40.8%	30.8%	28.4%
NMT	3.38	22.0%	28.4%	49.6%	2.90	35.6%	32.4%	32.0%
Ours	3.75	18.4%	12.8%	68.8%	3.55	18.8%	22.8%	58.4%

Table 11. Human Evaluation(Java dataset)

Model	Naturalness	Low_N	$Medium_N$	$High_N$	Relevance	Low_R	$Medium_R$	$High_R$
IR	3.56	19.6%	22.8%	57.6%	2.29	68.4%	14.4%	17.2%
Moses	2.37	62.4%	18.4%	19.2%	2.24	65.2%	21.6%	13.2%
NMT	2.96	28.0%	45.2%	26.8%	2.66	47.2%	27.6%	25.2%
Ours	3.42	22.0%	27.2%	50.8%	3.25	28.8%	24.4%	26.8%

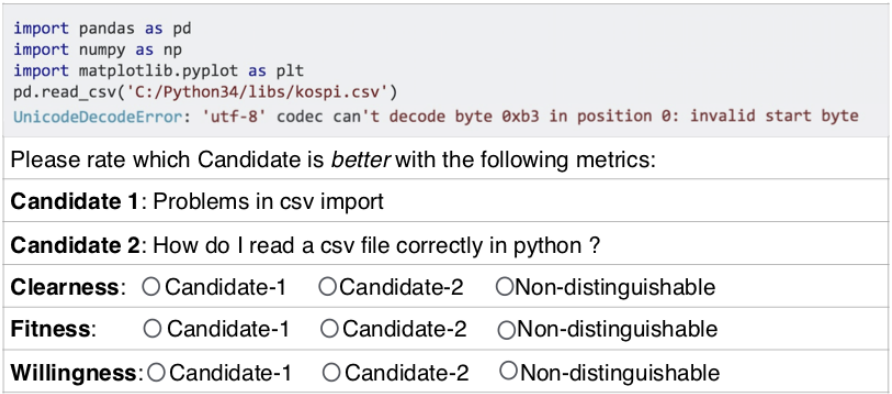

Fig. 10. User Study Case (Practical Manual Evaluation)

6.4 RQ-4: How effective is our approach for improving low-quality questions?

6.4.1 Practical Manual Evaluation Results. Fig. 10 shows one example of our practical manual evaluation study. We collected 50 pairs of question titles (one was generated by humans and one was generated by our approach) for Python and Java respectively for comparison purposes. For each pairwise comparison, we got 5 groups of selections from the evaluators. Each group contains three user selections with respect to the *Clearness*, *Fitness* and *Willingness* measures respectively. We calculated the proportion of the user selection according to each evaluation metric. Table 12 and Table 13 show the results of the practical manual evaluation for Python and Java respectively. From the table we can see that:

(1) The question titles generated by our approach outperform the poor quality question titles in terms of all the metrics. This demonstrates that our approach produces more clear and/or appropriate question titles, which is potentially helpful for improving the low-quality questions in Stack Overflow.
(2) Particularly, our question titles have substantially better willingness scores, indicating that developers are more willing to respond to our questions. This shows that question titles generated by our model are more likely to elicit further interactions, which is helpful to increase the likelihood of receiving answers.

Table 12. Practical Manual Evaluation (Python dataset)

Ours vs. Human	Win (%)	Lose (%)	Non-distinguishable (%)
Clearness	52.4	33.2	14.4
Fitness	55.2	24.0	20.8
Willingness	61.2	31.6	7.2

Table 13. Practical Manual Evaluation (Java dataset)

Ours vs. Human	Win (%)	Lose (%)	Non-distinguishable (%)
Clearness	42.8	34.0	23.2
Fitness	47.2	39.6	13.2
Willingness	49.2	26.8	24.0

6.4.2 Examples of Practical Manual Evaluation. Fig. 11 presents three examples of manual evaluation results. From these cases we can see that:

(1) The question titles with poor scores in Stack Overflow are often unclear (e.g., Example1) and/or unappropriate (e.g., Example2). For such cases, the question titles generated by our approach are more clear and attractive, such as Example1, and also questioning on key information. For example, the newly generated question titles in Example2 are much more appreciated by the evaluators than the original ones, which increases the likelihood and willingness of the developers to offer help.
(2) Not all of the poor quality question titles can be improved by our approach. Notably for some posts, our approach suffered from semantic drift, that is the questions generated by our approach do not align well with the developers' intent. Such as in Example3, the developer's problem was more about "writing with large data", while the semantics of our question generated has drifted to the problem of "java with bytebuffer". This is because the string variable "very large data" has been replaced by STR during data preprocessing, such information loss hinders the learning process of our approach.
(3) Even though the results generated by our approach are still not perfect, our approach is the first step on this topic and we also release our code and dataset to inspire further follow-up work.

Answer to *RQ-4:* How effective is our approach for improving low-quality questions? In general, for a large number of low-quality questions in Stack Overflow, our approach can improve the quality of the question titles via *Clearness*, *Fitness* and *Willingness* measures.

6.5 RQ-5: How effective is our use of *attention* mechanism, *copy* mechanism and *coverage* mechanism under automatic evaluation?

6.5.1 Ablation Analysis Results. We added an *attention* mechanism, a *copy* mechanism and a *coverage* mechanism to our sequence-to-sequence architecture. The ablation analysis is to verify the effectiveness of the three mechanisms, to be more specific, we compare our approach with several of its incomplete variants:

- **Model$_{\text{Atten+Copy}}$** removes the *coverage* mechanism from our approach.
- **Model$_{\text{Atten+Coverage}}$** removes the *copy* mechanism from our approach.
- **Model$_{\text{Atten}}$** removes the *copy* and *coverage* mechanism from our approach.
- **Model$_{\text{Basic}}$** removes all the *attention*, *copy* and *coverage* mechanism from our approach.

Example1(6795345) — Question Score: -3	**Example2** (4099140) — Question Score: -3	**Example3** (876602) — QuestionScore: -3

import urllib2, urllib
from BeautifulSoup import BeautifulSoup
import re
import urlparse
…
raw = urllib.rulopen(url)
soup = BeautifulSoup(raw)
parse = list(urlparse.urlparse(url))
 	
for ender in soup.findAll(ender):
links = "%(src)s"% ender
if ".jpg" in links: end = ".jpg"
if ".jpeg" in links: end = ".jpeg"
if ".gif" in links: end = ".gif"
if ".png" in links: end = ".png"
i += 1
urllib.urlretrieve(links, "%s%s" % (i, end))
	import org.xsocket.connection.*;
	import java.io.IOException;
	
	public class SocketClient {
	public static void main(String[] args) {
	try {
	IBlockingConnection bc = new
	BlockingConnection("127.0.0.1", 8090);
	String req = "Hello server";
	bc.write(req + "\r\n");
	} catch (IOException e) {}
	System.out.println("missing");
	}
	}
	
	C:\Users\Wildfire\Desktop>java -cp xSocket-2.8.14.jar
	SocketClient.java
	Exception in thread "main"
	java.lang.NoClassDefFoundError: SocketClient/java
		message = "very large data"+"\n";
		
		ByteBuffer buf = ByteBuffer.wrap(message.getBytes());
		
		int nbytes = channel.write(buf);

Human: Need help with a Python scraper	**Human**: Can compile but not run the code	**Human**: problem with writing large data using java nio socket channel
Ours: how to extract all links from url using beautiful soup?	**Ours**: java - how do i handle the noclassdeffounderror ?	**Ours**: how to write nbytes in java with bytebuffer ?
Ours vs. Human: Clearness(5:0) Fitness(4:1) Willingness(5:0)	**Ours vs. Human**: Clearness(4:1) Fitness(4:1) Willingness(5:0)	**Ours vs. Human**: Clearness(2:2) Fitness(1:4) Willingness(3:2)

Fig. 11. Practical Manual Evaluation Example

The ablation analysis results are presented in the Table 14 and Table 15. We can observe the following points:

(1) By comparing the results of $\textbf{Model}_{\textbf{Basic}}$ and $\textbf{Model}_{\textbf{Atten}}$, it is clear that incorporating an *attention* mechanism is able to improve the overall performance. For example, by adding *attention* mechanism, the average BLEU-4 score of the Attention-based model was improved by 9% and 13.3%, ROUGE-L score was improved by 6.8% and 10.8% on Python and Java dataset respectively. We attribute this to the ability of *attention* mechanism to perform better content selection, which can focus on the more salient part of the source code snippet.
(2) By comparing $\textbf{Model}_{\textbf{Atten}}$ with $\textbf{Model}_{\textbf{Atten+Copy}}$ and $\textbf{Model}_{\textbf{Atten+Coverage}}$, we can measure the performance improvements achieved due to the incorporation of *copy* mechanism and *coverage* mechanism respectively. Better performance can be achieved by solely adding *copy* or *coverage* mechanism to the attention-based model. This signals that both *copy* and *coverage* mechanism do have contributions to the performance improvements.
(3) Without *copy* mechanism, there is a drop overall in every evaluation measure, the ROUGE-L score drops 13% and 9.4% on Python and Java dataset respectively. On the other hand, without *coverage* mechanism, we see a consistent but sufficiently lower drop in each evaluation measure, the ROUGE-L drops 12.3% on Python and 3.8% on Java.
(4) By comparing the results of our approach with each of the variant model, we can see that no matter which type of mechanism we dropped, it does hurt the performance of our model. This verifies the importance and effectiveness of these three mechanisms.

6.5.2 Examples of Ablation Analysis. To gain further insight into our approach, we further illustrate some examples from the ablation analysis to show the effect of employing the *attention*, *copy* and *coverage* mechanism. The results are shown in Fig. 12, we can see that:

(1) Question titles generated by the basic model are of low-quality. Comparing the results of the basic model and attention model, we can see that by adding the *attention* mechanism, the generated question titles are more meaningful and relevant for the given code snippet. The *attention* mechanism enables the model to focus on the relevant parts of the input sequence as needed. As shown in Example1, the model will focus on the "request" related segment in source code when it generates "post request" for the question title.

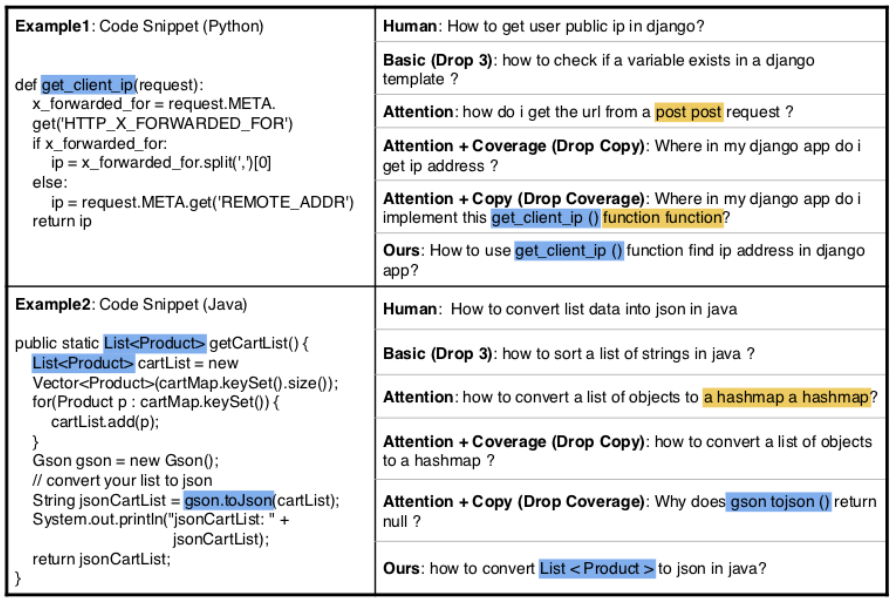

Fig. 12. Ablation Analysis Example

(2) Repetition is a common problem for attentional sequence to sequence models (e.g., [53, 56, 59]). Meaningless repeated words are produced during the generation process (highlighted with yellow color). We introduce a *coverage* mechanism for discouraging such repetitions in our generator by quantitatively emphasizing the coverage of sentence words while decoding. As can be seen in Example2, "a harshmap" has been meaningless repeated twice, employing the *coverage* mechanism can effectively discourage such repetitions.

(3) We observe that a *high-quality* question title is generated using our approach. Recall that a code snippet usually contains tokens (highlighted with a blue color) with very rare occurrences. It is difficult for a decoder to generate such words solely based on language modeling. For such cases, we incorporate the *copy* mechanism to copy the rare tokens from the code snippet to the question title. In the first example, the method name *get_client_ip* has been properly picked up from the source code snippet to the generated question titles.

Answer to *RQ-5:* How effective is our use of *attention* mechanism, *copy* mechanism and *coverage* mechanism under automatic evaluation? In summary, all the three mechanisms, i.e., *attention* mechanism, *copy* mechanism, *coverage* mechanism, are effective and helpful to enhance the performance of our approach.

6.6 RQ-6: How effective is our approach under different parameter settings?

One of the key parameter of our approach is the vocabulary size. The encoder-decoder architecture models need a fixed vocabulary for the source input and target output. To generate all the possible words, the basic Seq2Seq model has to include all the vocabulary tokens that appeared in the training set, which requires a lot of time and memory to train the models. One advantage of our model is that, with the help of *copy* mechanism, our approach can copy words from source input to the target output. We can maintain a small size vocabulary which exclude the low frequency words, but also get better performance and generalization ability.

Table 14. Ablation evaluation (Python dataset)

Measure	$\text{Model}_{\text{Basic}}$	$\text{Model}_{\text{Atten}}$	$\text{Model}_{\text{Atten+Coverage}}$	$\text{Model}_{\text{Atten+Copy}}$	Ours
BLEU-1	25.1 ± 1.5%	28.6 ± 1.7%	29.6 ± 1.8%	31.5 ± 1.9%	35.8 ± 2.0%
BLEU-2	20.2 ± 0.7%	22.3 ± 0.8%	24.6 ± 0.6%	27.8 ± 0.8%	30.1 ± 0.9%
BLEU-3	19.1 ± 0.4%	21.7 ± 0.4%	23.8 ± 0.5%	25.4 ± 0.4%	26.8 ± 0.4%
BLEU-4	18.7 ± 0.3%	20.3 ± 0.3%	22.3 ± 0.2%	23.1 ± 0.2%	24.2 ± 0.3%
ROUGE-1	32.8 ± 2.0%	34.1 ± 2.3%	35.3 ± 2.2%	35.4 ± 2.4%	39.9 ± 2.5%
ROUGE-2	9.1 ± 0.8%	10.2 ± 1.2%	10.6 ± 2.1%	10.8 ± 2.0%	12.6 ± 2.5%
ROUGE-L	29.2 ± 5.8%	31.2 ± 2.0%	31.9 ± 2.1%	32.2 ± 2.2%	36.7 ± 2.4%

Table 15. Ablation evaluation (Java dataset)

Measure	$\text{Model}_{\text{Basic}}$	$\text{Model}_{\text{Atten}}$	$\text{Model}_{\text{Atten+Coverage}}$	$\text{Model}_{\text{Atten+Copy}}$	Ours
BLEU-1	20.5 ± 1.0%	25.2 ± 1.6%	27.8 ± 1.6%	29.7 ± 1.7%	31.8 ± 1.8%
BLEU-2	16.4 ± 0.6%	20.7 ± 0.7%	25.0 ± 0.6%	26.1 ± 0.6%	27.5 ± 0.7%
BLEU-3	17.8 ± 0.4%	21.1 ± 0.3%	23.6 ± 0.3%	24.4 ± 0.3%	25.2 ± 0.3%
BLEU-4	18.1 ± 0.2%	20.5 ± 0.2%	22.0 ± 0.1%	22.6 ± 0.2%	23.3 ± 0.2%
ROUGE-1	28.3 ± 1.3%	30.5 ± 2.0%	31.2 ± 2.0%	33.2 ± 2.1%	35.4 ± 2.2%
ROUGE-2	6.9 ± 0.5%	7.9 ± 1.1%	8.2 ± 1.2%	8.7 ± 1.5%	10.0 ± 1.8%
ROUGE-L	24.6 ± 1.1%	27.3 ± 1.8%	28.8 ± 1.9%	30.6 ± 2.0%	31.8 ± 2.2%

Table 16. Vocab Size & Training Time(per epoch)

	Threshold	Vocab Size	Training Time
Python	1	58,536	766.9
	2	49,656	719.1
	3	36,277	663.7
	5	22,244	593.8
	7	16,368	549.2
	10	12,142	539.1
	100	2,503	499.9
	Threshold	Vocab Size	Training Time
Java	1	221,160	2218.3
	2	131,862	1692.3
	3	79,048	1074.1
	5	54,352	962.2
	7	38,670	898.4
	10	27,341	831.4
	100	4,642	723.8

We set different word frequency threshold, i.e., 1, 2, 3, 5, 7, 10, 100, for constructing the vocabulary. Setting word frequency threshold to 1 means the vocabulary is constructed with words that appeared at least twice in the training set. Different models were trained under these parameters on the Python and Java datasets separately. The vocabulary size and training time under different threshold are summarised in Table 16. Fig. 13 and Fig. 14 shows the influence of different threshold settings on the BLEU-4 score and ROUGE-L score. We have the following observations from these figures:

(1) Our approach achieves its best performance on Java dataset when the similarity threshold set to 3, the corresponding vocabulary size is 79,048. When the vocabulary size is too big,

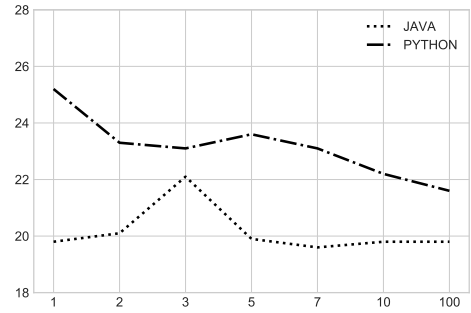

Fig. 13. BLEU4 Score under different vocab threshold

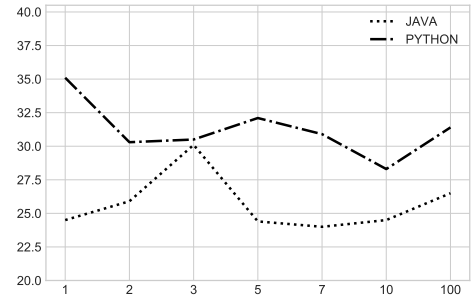

Fig. 14. ROUGE-L Score under different vocab threshold

i.e., 221,160 with threshold equals 1, the BLEU4 and ROUGE-L score becomes lower. This is because some non-generic words will be included in the fixed vocabulary, which leads to difficulties for our approach to learn how to copy words from the input source sequence.

(2) The results of our approach are best on Python dataset when the word frequency threshold set to 1, the corresponding vocabulary size is 58,536. Compared with the results of the Java dataset, the optimum vocabulary size settings of our approach can be around 60000.

(3) When the word frequency threshold rockets up to 100, the vocabulary size decreases to 2,503 and 4,626 on Python and Java dataset respectively. Even with a much smaller vocabulary size, our approach can still have a comparable performance against Basic Seq2Seq model, which further supports the generalization ability of our approach.

Another parameter of our approach is the dimension of word embeddings. We choose five different word embedding sizes, i.e., 100, 200, 300, 400, 500, and qualitatively compare the performance of our approach in these different word embeddings. Fig. 15 and Fig. 16 show the influence of different word embedding sizes on the BLEU-4 and ROUGE-L score. One can clearly see that our approach achieves the best BLEU-4 and ROUGE-L score when the embedding size is set to 300. Too large word embedding size may not be helpful to improve the accuracy.

6.7 RQ-7: How efficient is our approach in practical usage?

The experiment was conducted on an Nvidia GeForce GTX 1080 GPU with 8GB memory. The time cost of our approach is mostly for the training process which takes approximately 8 to 10 hours for

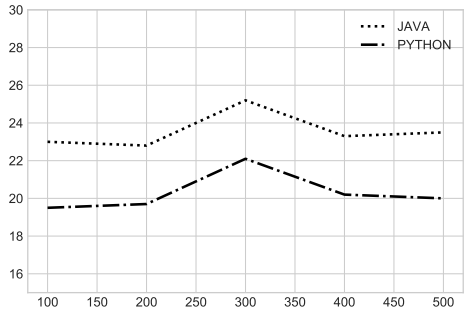

Fig. 15. BLEU4 Score under different sizes of word embeddings

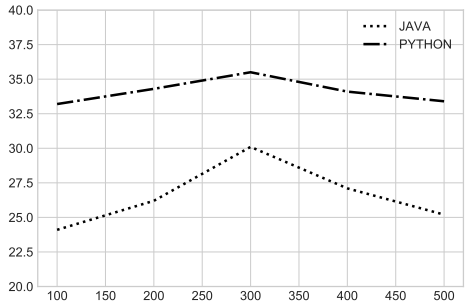

Fig. 16. ROUGE-L Score under different sizes of word embeddings

different datasets. The testing process on around 3,000 examples takes one to three minutes, while generating a single question title only costs 20 to 60ms.

Considering that the query for generating a question title using our approach is efficient, we have implemented our approach as a standalone web-based tool, named CODE2QUE, to facilitate developers in using our approach and to inspire follow up research. Fig. 17 shows the web interface of CODE2QUE. Developers can copy and paste their code snippet into our web application. CODE2QUE embeds the code snippet via source code encoder and generates the question titles for the developers. We below describe the details of the input and output of such a process.

- **Input:** the input to the CODE2QUE is a code snippet, which is an ordered sequence of source code lines. We have provided support for different types of programming languages (e.g., Python, Java, Javascript, C# and SQL) for users to select. The input box in Fig. 17 shows an example of a Python code snippet. After inputting the code snippet, the developers can click the "Generate" button to submit their query.
- **Output:** the output of CODE2QUE is in two parts: (i) Generated Questions: CODE2QUE will generate a question title using our backend model according to the code snippet and programming language they choose. For example, "*how to extract text from html pages using html2text*" is generated for the given code snippet. (ii) Retrieved Questions: After the developer submits his/her code snippet to the server, the code snippet is converted into a vector by our backend Source Code Encoder, then CODE2QUE searches through our codebase and returns the top3 questions with similar code snippets. The link to these questions on the Stack Overflow website is also provided for reference. Developers can use these to quickly browse the related questions to have a better understanding of the problem.

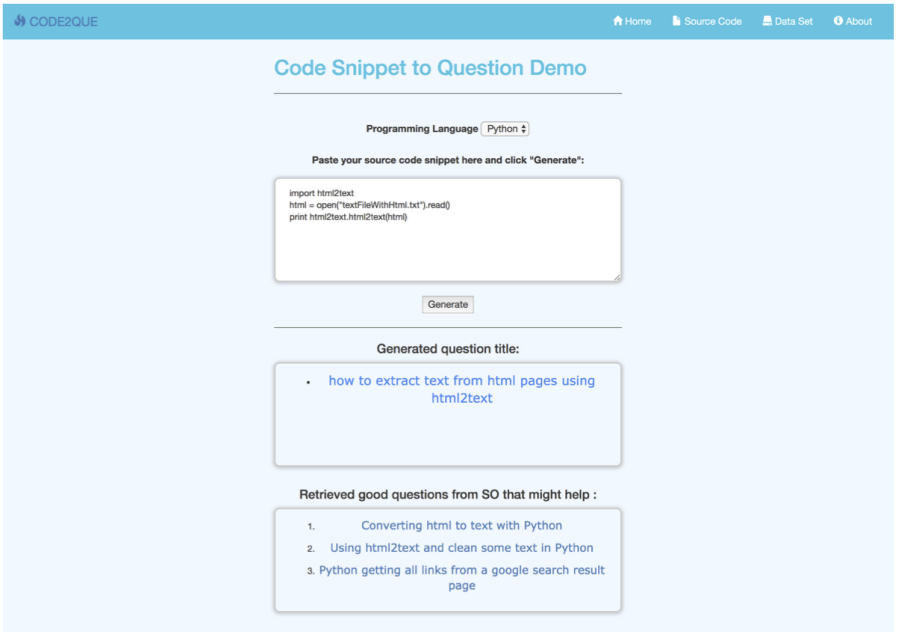

Fig. 17. CODE2QUE Web Service Tool

Answer to *RQ-7:* How efficient is our approach in practical usage? In summary, our approach is efficient enough for practical use and we have implemented a web service tool, named CODE2QUE, to apply our approach for practical use.

7 DISCUSSION

In this section, we discuss the main contribution of our work and analyze the strength and potential weakness of our work associated with each contribution.

7.1 Question Quality Improvement

It is important for CQA forums to maintain a satisfactory quality level for the questions and answers so as to improve community reputation and provide better user experience. Questions are a fundamental aspect of a CQA website. Poorly formulated questions are less likely to receive useful responses, thus hindering the overall knowledge generation and sharing process.

- *Strength of our work.* Previous work related to CQA quality studies focus on question quality prediction. For example, the authors in [51] developed a model for predicting question quality using the content of the question. The authors in [4] proposed a method to identify inappropriate questions by using previously asked similar questions. Different from the existing research, our study aims to improve low-quality questions in Stack Overflow. To the best of our knowledge, this is the first work that investigates the possibility of automatically improving low-quality questions in Stack Overflow.
- *Weakness of our work.* According to our practical manual evaluation results, our approach can improve a large number of low-quality questions in Stack Overflow via *Clearness*, *Fitness* and *Willingness* measures. However, the results generated by our approach are still not perfect,

and for some posts, our approach suffers from semantic drift problems. We plan to incorporate more context information for generating better question titles in the future.

7.2 Deep Sequence to Sequence Approach

Recently, deep learning has achieved promising results in solving many software engineering tasks, such as code search (e.g., [24, 31, 39]), code summarization (e.g., [30, 32, 33, 62]), and API recommendation (e.g., [25, 26]). Among these works, a number of researchers have applied the sequence to sequence methods for mining the ⟨natural language, code snippet⟩ pairs, such as the commit message generation. (e.g., [32, 33]).

- *Strength of our work.* A major challenge for question generation tasks in our study is the semantic gap between the code snippet and natural language descriptions. To bridge the gap between code fragment and natural language queries, we employed a deep sequence to sequence approach to build the neural language model for both code snippets and natural language questions. The neural language model automatically learns common patterns from the large scale source code snippets. Furthermore, different from the existing sequence to sequence learning approach, we add *attention*, *copy* and *coverage* mechanism to our sequence-to-sequence architecture to suit our specific task. The *attention* mechanism can perform better content selection from the input, while the *copy* mechanism can handle the rare word problems among the code snippet, and the *coverage* mechanism can eliminate the meaningless repetitions.
- *Weakness of our work.* Previous works [30, 32, 62] have shown that incorporating structural information of the source code (i.e., the AST) can improve the performance of the model, However, considering that the majority of the code snippets are not parsable in Stack Overflow, we do not use the AST structural information at the current stage. We plan to use the program repair algorithm to fix the code snippet in Stack Overflow and employ more contextual information of the source code in the future.

7.3 Question Generation Task

Stack Overflow is a collaborative question answering website, its target audience are software developers, maintenance professionals and programmers. Over the recent years, Stack Overflow has attracted increasing attention from the software engineering research community. However, since the questions and answers posted by developers on Stack Overflow are usually unstructured natural language texts containing code snippets, which makes it more challenging for researchers to mine and analyze these posts.

- *Strength of our work.* To improve the software development process, researchers have investigated the Stack Overflow knowledge-base for various software development activities, such as predicting the post quality [4, 50, 51, 72, 73], answer recommendation [22, 55, 70], code/questions retrieval [2, 9, 16, 29, 71] etc. However, to the best of our knowledge, this is the first work which investigates the question generation task in Stack Overflow. We first perform such a task to assist developers to generate a question title when presenting a code snippet.
- *Weakness of our work.* We collected more than 1M ⟨*code snippet, question*⟩ pairs from Stack Overflow, which covers a variety of programming languages (e.g., Python, Java, Javascript, C# and SQL). Considering our study is the first step on this topic, we have published our data to inspire further follow-up work. However, even though we have cleaned the data via pre-processing, some data may still be noisy. We plan to improve the dataset quality by further manual checking in the future.

8 THREATS TO VALIDITY

We have identified the following threats to validity among our study:

Internal Validity Threats to internal validity are concerned with potential errors in our code implementation and study settings. For the automatic evaluation, in order to reduce errors, we have double-checked and fully tested our source code. We have carefully tuned the parameters of the baseline approaches and used them in their highest performing settings for comparison, but there may still exist errors that we did not note. Considering such cases, we have published our source code and dataset to facilitate other researchers to replicate and extend our work.

External Validity The external validity relates to the quality and generalizability of our dataset. Our dataset is constructed from the official Stack Overflow data dump which contains a variety of programming languages, such as Python, Java, Javascript, C# and SQL. However, there are still many other programming languages in Stack Overflow which are not considered in our study. We believe that our results will generalize to other programming languages, due to the overall reasonable similarity in code snippets despite particular language syntax, semantics and APIs. We will try to extend our approach to other programming languages to benefit more users in future studies.

Construct Validity The construct validity concerns the relation between theory and observation. In this study, such threats are mainly due to the suitability of our evaluation measures. For the practical manual evaluation, the manual validation could be affected by the subjectiveness of the evaluators and the human errors. For the human evaluation, the evaluators' degree of carefulness, effort and English skills in the examination process may affect the validity of judgements. We minimized such threats by choosing experienced participants who have at least one year of studying/working experience in English speaking countries, and are familiar with Python and Java programming languages. We also gave the participants enough time to complete the evaluation tasks.

Conclusion Validity The conclusion validity relates to issues that could affect the ability to draw correct conclusions about relations between the treatment and the outcome of an experiment. One issue during the data filtering procedure is that we only keep the questions which contain several keywords, such as "how", "what", "why". However, since the questions in Stack Overflow can be rather complicated, our results do not shed light on how effective our solution is on questions of other kinds. On the other hand, from the human evaluation analysis, we see a key challenge for our current work is that the questions generated by our approach suffered from semantic drift. This is because it is difficult to judge a question poster's intent by solely looking at his/her code snippet. In such a case, more relevant information such as question description, question tags could further be incorporated within our model, which may help to generate a question that is more accurate and precise.

9 CONCLUSION AND FUTURE WORK

In this work, we have proposed a model for the task of automatic question generation based on a given code snippet. Our model is based on sequence-to-sequence architecture, and enhanced with an *attention* mechanism to perform better content selection, a *copy* mechanism to handle the rare-words problem within the input code snippet as well as *coverage* mechanism to discourage the meaningless repetitions. We carried out comprehensive evaluation on Stack Overflow datasets to demonstrate the effectiveness of our approach, compared with several existing baselines, our model achieves the best performance in both the automatic evaluation and human evaluation. We have also released our code and datasets to facilitate other researchers to verify their ideas and inspire

the follow up work. For future work, we plan to design better models to generate meaningful question titles by considering extra context information, such as question description. Additional work will be needed to address this context-sensitive question generation task.

10 ACKNOWLEDGEMENTS

This research was partially supported by the Australian Research Council's Discovery Early Career Researcher Award (DECRA) funding scheme (DE200100021), ARC Laureate Fellowship funding scheme (FL190100035), and ARC Discovery grant DP170101932.

REFERENCES

[1] Miltiadis Allamanis and Charles Sutton. 2013. Why, when, and what: analyzing stack overflow questions by topic, type, and code. In *Proceedings of the 10th Working Conference on Mining Software Repositories*. IEEE Press, 53–56.

[2] Miltos Allamanis, Daniel Tarlow, Andrew Gordon, and Yi Wei. 2015. Bimodal modelling of source code and natural language. In *International Conference on Machine Learning*. 2123–2132.

[3] Ashton Anderson, Daniel Huttenlocher, Jon Kleinberg, and Jure Leskovec. 2012. 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*. 850–858.

[4] Piyush Arora, Debasis Ganguly, and Gareth JF Jones. 2015. 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, 1232–1239.

[5] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473* (2014).

[6] Steven Bird and Edward Loper. 2004. NLTK: the natural language toolkit. In *Proceedings of the ACL 2004 on Interactive poster and demonstration sessions*. Association for Computational Linguistics, 31.

[7] Lutz Büch and Artur Andrzejak. 2019. Learning-based recursive aggregation of abstract syntax trees for code clone detection. In *2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 95–104.

[8] Fabio Calefato, Filippo Lanubile, and Nicole Novielli. 2018. How to ask for technical help? Evidence-based guidelines for writing questions on Stack Overflow. *Information and Software Technology* 94 (2018), 186–207.

[9] Guibin Chen, Chunyang Chen, Zhenchang Xing, and Bowen Xu. 2016. Learning a dual-language vector space for domain-specific cross-lingual question retrieval. In *2016 31st IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 744–755.

[10] Zimin Chen, Steve James Kommrusch, Michele Tufano, Louis-Noël Pouchet, Denys Poshyvanyk, and Martin Monperrus. 2019. Sequencer: Sequence-to-sequence learning for end-to-end program repair. *IEEE Transactions on Software Engineering* (2019).

[11] Denzil Correa and Ashish Sureka. 2013. Fit or unfit: analysis and prediction of'closed questions' on stack overflow. In *Proceedings of the first ACM conference on Online social networks*. ACM, 201–212.

[12] Denzil Correa and Ashish Sureka. 2014. Chaff from the wheat: Characterization and modeling of deleted questions on stack overflow. In *Proceedings of the 23rd international conference on World wide web*. 631–642.

[13] Aditya Desai, Sumit Gulwani, Vineet Hingorani, Nidhi Jain, Amey Karkare, Mark Marron, Subhajit Roy, et al. 2016. Program synthesis using natural language. In *Proceedings of the 38th International Conference on Software Engineering*. ACM, 345–356.

[14] Maarten Duijn, Adam Kucera, and Alberto Bacchelli. 2015. Quality questions need quality code: Classifying code fragments on stack overflow. In *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*. IEEE, 410–413.

[15] Christine Franks, Zhaopeng Tu, Premkumar Devanbu, and Vincent Hellendoorn. 2015. Cacheca: A cache language model based code suggestion tool. In *Proceedings of the 37th International Conference on Software Engineering-Volume 2*. IEEE Press, 705–708.

[16] Debasis Ganguly and Gareth JF Jones. 2015. Partially labeled supervised topic models for RetrievingSimilar questions in CQA forums. In *Proceedings of the 2015 International Conference on The Theory of Information Retrieval*. ACM, 161–170.

[17] Zhipeng Gao. 2020. Dataset for the paper: Generating Question Titles for Stack Overflow from Mined Code Snippets. https://doi.org/10.5281/zenodo.3816592

[18] Zhipeng Gao, Vinoj Jayasundara, Lingxiao Jiang, Xin Xia, David Lo, and John Grundy. 2019. SmartEmbed: A Tool for Clone and Bug Detection in Smart Contracts through Structural Code Embedding. In *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 394–397.

[19] Z. Gao, L. Jiang, X. Xia, D. Lo, and J. Grundy. 2020. Checking Smart Contracts with Structural Code Embedding. *IEEE Transactions on Software Engineering* (2020), 1–1. https://doi.org/10.1109/TSE.2020.2971482

[20] Alessandra Giordani and Alessandro Moschitti. 2009. Semantic mapping between natural language questions and SQL queries via syntactic pairing. In *International Conference on Application of Natural Language to Information Systems*. Springer, 207–221.

[21] Alessandra Giordani and Alessandro Moschitti. 2012. Translating questions to SQL queries with generative parsers discriminatively reranked. *Proceedings of COLING 2012: Posters* (2012), 401–410.

[22] George Gkotsis, Karen Stepanyan, Carlos Pedrinaci, John Domingue, and Maria Liakata. 2014. It's all in the content: state of the art best answer prediction based on discretisation of shallow linguistic features. In *Proceedings of the 2014 ACM conference on Web science*. ACM, 202–210.

[23] Jiatao Gu, Zhengdong Lu, Hang Li, and Victor OK Li. 2016. Incorporating copying mechanism in sequence-to-sequence learning. *arXiv preprint arXiv:1603.06393* (2016).

[24] Xiaodong Gu, Hongyu Zhang, and Sunghun Kim. 2018. Deep code search. In *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)*. IEEE, 933–944.

[25] Xiaodong Gu, Hongyu Zhang, Dongmei Zhang, and Sunghun Kim. 2016. Deep API learning. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*. 631–642.

[26] Xiaodong Gu, Hongyu Zhang, Dongmei Zhang, and Sunghun Kim. 2017. DeepAM: Migrate APIs with multi-modal sequence to sequence learning. *arXiv preprint arXiv:1704.07734* (2017).

[27] Sumit Gulwani and Mark Marron. 2014. Nlyze: Interactive programming by natural language for spreadsheet data analysis and manipulation. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*. ACM, 803–814.

[28] Kenneth Heafield. 2011. KenLM: Faster and smaller language model queries. In *Proceedings of the Sixth Workshop on Statistical Machine Translation*. Association for Computational Linguistics, 187–197.

[29] Stefan Henβ, Martin Monperrus, and Mira Mezini. 2012. Semi-automatically extracting FAQs to improve accessibility of software development knowledge. In *Proceedings of the 34th International Conference on Software Engineering*. IEEE Press, 793–803.

[30] Xing Hu, Ge Li, Xin Xia, David Lo, and Zhi Jin. 2018. Deep code comment generation. In *Proceedings of the 26th Conference on Program Comprehension*. ACM, 200–210.

[31] Hamel Husain, Ho-Hsiang Wu, Tiferet Gazit, Miltiadis Allamanis, and Marc Brockschmidt. 2019. CodeSearchNet Challenge: Evaluating the State of Semantic Code Search. *arXiv preprint arXiv:1909.09436* (2019).

[32] Srinivasan Iyer, Ioannis Konstas, Alvin Cheung, and Luke Zettlemoyer. 2016. 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)*, Vol. 1. 2073–2083.

[33] Siyuan Jiang, Ameer Armaly, and Collin McMillan. 2017. Automatically generating commit messages from diffs using neural machine translation. In *Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering*. IEEE Press, 135–146.

[34] Xianhao Jin and Francisco Servant. 2019. 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, 225–229.

[35] Iman Keivanloo, Juergen Rilling, and Ying Zou. 2014. Spotting working code examples. In *Proceedings of the 36th International Conference on Software Engineering*. ACM, 664–675.

[36] Philipp Koehn, Hieu Hoang, Alexandra Birch, Chris Callison-Burch, Marcello Federico, Nicola Bertoldi, Brooke Cowan, Wade Shen, Christine Moran, Richard Zens, et al. 2007. Moses: Open source toolkit for statistical machine translation. In *Proceedings of the 45th annual meeting of the ACL on interactive poster and demonstration sessions*. Association for Computational Linguistics, 177–180.

[37] Baichuan Li, Tan Jin, Michael R Lyu, Irwin King, and Barley Mak. 2012. Analyzing and predicting question quality in community question answering services. In *Proceedings of the 21st International Conference on World Wide Web*. 775–782.

[38] Fei Li and Hosagrahar V Jagadish. 2014. NaLIR: an interactive natural language interface for querying relational databases. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*. ACM, 709–712.

[39] Hongyu Li, Seohyun Kim, and Satish Chandra. 2019. Neural Code Search Evaluation Dataset. *arXiv preprint arXiv:1908.09804* (2019).

[40] Chin-Yew Lin. 2004. Rouge: A package for automatic evaluation of summaries. *Text Summarization Branches Out* (2004).

[41] Wang Ling, Edward Grefenstette, Karl Moritz Hermann, Tomáš Kočiskỳ, Andrew Senior, Fumin Wang, and Phil Blunsom. 2016. Latent predictor networks for code generation. *arXiv preprint arXiv:1603.06744* (2016).

[42] Jing Liu, Quan Wang, Chin-Yew Lin, and Hsiao-Wuen Hon. 2013. Question difficulty estimation in community question answering services. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*. 85–90.
[43] Nicholas Locascio, Karthik Narasimhan, Eduardo DeLeon, Nate Kushman, and Regina Barzilay. 2016. Neural generation of regular expressions from natural language with minimal domain knowledge. *arXiv preprint arXiv:1608.03000* (2016).
[44] Lena Mamykina, Bella Manoim, Manas Mittal, George Hripcsak, and Björn Hartmann. 2011. Design lessons from the fastest q&a site in the west. In *Proceedings of the SIGCHI conference on Human factors in computing systems*. 2857–2866.
[45] Ali Mesbah, Andrew Rice, Emily Johnston, Nick Glorioso, and Edward Aftandilian. 2019. DeepDelta: learning to repair compilation errors. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 925–936.
[46] Seyed Mehdi Nasehi, Jonathan Sillito, Frank Maurer, and Chris Burns. 2012. 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, 25–34.
[47] Liqiang Nie, Xiaochi Wei, Dongxiang Zhang, Xiang Wang, Zhipeng Gao, and Yi Yang. 2017. Data-driven answer selection in community QA systems. *IEEE transactions on knowledge and data engineering* 29, 6 (2017), 1186–1198.
[48] Yusuke Oda, Hiroyuki Fudaba, Graham Neubig, Hideaki Hata, Sakriani Sakti, Tomoki Toda, and Satoshi Nakamura. 2015. Learning to generate pseudo-code from source code using statistical machine translation (t). In *Automated Software Engineering (ASE), 2015 30th IEEE/ACM International Conference on*. IEEE, 574–584.
[49] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. BLEU: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting on association for computational linguistics*. Association for Computational Linguistics, 311–318.
[50] Luca Ponzanelli, Andrea Mocci, Alberto Bacchelli, and Michele Lanza. 2014. Understanding and classifying the quality of technical forum questions. In *Quality Software (QSIC), 2014 14th International Conference on*. IEEE, 343–352.
[51] Sujith Ravi, Bo Pang, Vibhor Rastogi, and Ravi Kumar. 2014. Great question! question quality in community q&a. In *Eighth International AAAI Conference on Weblogs and Social Media*.
[52] Stephen E Robertson and Steve Walker. 1994. Some simple effective approximations to the 2-poisson model for probabilistic weighted retrieval. In *Proceedings of the 17th annual international ACM SIGIR conference on Research and development in information retrieval*. Springer-Verlag New York, Inc., 232–241.
[53] Baskaran Sankaran, Haitao Mi, Yaser Al-Onaizan, and Abe Ittycheriah. 2016. Temporal attention model for neural machine translation. *arXiv preprint arXiv:1608.02927* (2016).
[54] Sanja Seljan, Marija Brkic, and Tomislav Vicic. 2012. BLEU Evaluation of Machine-Translated English-Croatian Legislation.. In *LREC*. 2143–2148.
[55] Priyanka Singh and Elena Simperl. 2016. Using semantics to search answers for unanswered questions in q&a forums. In *Proceedings of the 25th International Conference Companion on World Wide Web*. International World Wide Web Conferences Steering Committee, 699–706.
[56] Jun Suzuki and Masaaki Nagata. 2016. Rnn-based encoder-decoder approach with word frequency estimation. *arXiv preprint arXiv:1701.00138* (2016).
[57] Laszlo Pal Toth, Balázs Nagy, Dávid Janthó, László Vidács, and Tibor Gyimóthy. 2019. Towards an Accurate Prediction of the Question Quality on Stack Overflow using a Deep-Learning-Based NLP Approach. In *ICSOFT*.
[58] Jan Trienes and Krisztian Balog. 2019. Identifying Unclear Questions in Community Question Answering Websites. In *ECIR*.
[59] Zhaopeng Tu, Zhengdong Lu, Yang Liu, Xiaohua Liu, and Hang Li. 2016. Modeling coverage for neural machine translation. *arXiv preprint arXiv:1601.04811* (2016).
[60] Marko Vasic, Aditya Kanade, Petros Maniatis, David Bieber, and Rishabh Singh. 2019. Neural program repair by jointly learning to localize and repair. *arXiv preprint arXiv:1904.01720* (2019).
[61] Venkatesh Vinayakarao, Anita Sarma, Rahul Purandare, Shuktika Jain, and Saumya Jain. 2017. Anne: Improving source code search using entity retrieval approach. In *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining*. ACM, 211–220.
[62] Yao Wan, Zhou Zhao, Min Yang, Guandong Xu, Haochao Ying, Jian Wu, and Philip S Yu. 2018. Improving automatic source code summarization via deep reinforcement learning. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*. ACM, 397–407.
[63] Shaowei Wang, David Lo, Bogdan Vasilescu, and Alexander Serebrenik. 2018. EnTagRec++: An enhanced tag recommendation system for software information sites. *Empirical Software Engineering* 23, 2 (2018), 800–832.
[64] Wenhan Wang, Ge Li, Bo Ma, Xin Xia, and Zhi Jin. 2020. Detecting Code Clones with Graph Neural Networkand Flow-Augmented Abstract Syntax Tree. *arXiv preprint arXiv:2002.08653* (2020).
[65] Xin-Yu Wang, Xin Xia, and David Lo. 2015. Tagcombine: Recommending tags to contents in software information sites. *Journal of Computer Science and Technology* 30, 5 (2015), 1017–1035.

[66] Martin White, Michele Tufano, Matias Martinez, Martin Monperrus, and Denys Poshyvanyk. 2019. Sorting and transforming program repair ingredients via deep learning code similarities. In *2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 479–490.

[67] Martin White, Michele Tufano, Christopher Vendome, and Denys Poshyvanyk. 2016. Deep learning code fragments for code clone detection. In *2016 31st IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 87–98.

[68] Edmund Wong, Jinqiu Yang, and Lin Tan. 2013. Autocomment: Mining question and answer sites for automatic comment generation. In *Automated Software Engineering (ASE), 2013 IEEE/ACM 28th International Conference on*. IEEE, 562–567.

[69] Xin Xia, David Lo, Xinyu Wang, and Bo Zhou. 2013. Tag recommendation in software information sites. In *2013 10th Working Conference on Mining Software Repositories (MSR)*. IEEE, 287–296.

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

[71] Bowen Xu, Zhenchang Xing, Xin Xia, David Lo, and Shanping Li. 2018. Domain-specific cross-language relevant question retrieval. *Empirical Software Engineering* 23, 2 (2018), 1084–1122.

[72] Jie Yang, Claudia Hauff, Alessandro Bozzon, and Geert-Jan Houben. 2014. Asking the right question in collaborative q&a systems. In *Proceedings of the 25th ACM conference on Hypertext and social media*. ACM, 179–189.

[73] Yuan Yao, Hanghang Tong, Tao Xie, Leman Akoglu, Feng Xu, and Jian Lu. 2013. Want a good answer? ask a good question first! *arXiv preprint arXiv:1311.6876* (2013).

[74] Pengcheng Yin and Graham Neubig. 2017. A syntactic neural model for general-purpose code generation. *arXiv preprint arXiv:1704.01696* (2017).

[75] Tianyi Zhang, Ganesha Upadhyaya, Anastasia Reinhardt, Hridesh Rajan, and Miryung Kim. 2018. 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, 886–896.

[76] Jiangang Zhu, Beijun Shen, Xuyang Cai, and Haofen Wang. 2015. Building a Large-scale Software Programming Taxonomy from Stackoverflow.. In *SEKE*. 391–396.

Received Mar 2019; revised ?? 2019; accepted ?? 2019