Code2Que: A Tool for Improving Question Titles from Mined Code Snippets in Stack Overflow

Zhipeng Gao
Monash University, Australia

Xin Xia
Monash University, Australia

David Lo
Singapore Management University,
Singapore

John Grundy
Monash University, Australia

Yuan-Fang Li
Monash University, Australia

ABSTRACT

Stack Overflow is one of the most popular technical Q&A sites used by software developers. Seeking help from Stack Overflow has become an essential part of software developers' daily work for solving programming-related questions. Although the Stack Overflow community has provided quality assurance guidelines to help users write better questions, we observed that a significant number of questions submitted to Stack Overflow are of low quality. In this paper, we introduce a new web-based tool, Code2Que, which can help developers in writing higher quality questions for a given code snippet. Code2Que consists of two main stages: offline learning and online recommendation. In the offline learning phase, we first collect a set of good quality ⟨*code snippet, question*⟩ pairs as training samples. We then train our model on these training samples via a deep sequence-to-sequence approach, enhanced with an *attention* mechanism, a *copy* mechanism and a *coverage* mechanism. In the online recommendation phase, for a given code snippet, we use the offline trained model to generate question titles to assist less experienced developers in writing questions more effectively. To evaluate Code2Que, we first sampled 50 low quality ⟨*code snippet, question*⟩ pairs from the Python and Java datasets on Stack Overflow. Then we conducted a user study to evaluate the question titles generated by our approach as compared to human-written ones using three metrics: *Clearness*, *Fitness* and *Willingness to Respond*. Our experimental results show that for a large number of low-quality questions in Stack Overflow, Code2Que can improve the question titles in terms of *Clearness*, *Fitness* and *Willingness* measures. Code2Que can be accessed at http://www.code2que.com. A demo video of Code2Que is at https://youtu.be/orG--uXKnkU.

1 INTRODUCTION

In recent years, Stack Overflow (SO) has become one of the most common ways that developers seek programming problem-related answers on the web. Millions of developers now use Stack Overflow to search for high-quality posts to solve their daily work problems. The success of Stack Overflow and community-based question and answer sites in general rely heavily on the will of community members to answer other's questions. Intuitively, a well-phrased question is more likely to obtain attention from potential experts, thus increasing the likelihood of receiving useful help and support. In contrast, poorly asked questions may discourage potential helpers and are less likely to receive useful answers, or indeed any answer at all.

Even though Stack Overflow has provided detailed guidelines to help community members post well-written questions, a large number of questions submitted to Stack Overflow are of low quality. These poorly asked questions are, more often than not, ambiguous, vague, and/or incomplete. It is thus very hard to attract potential experts to provide useful answers, which may discourage the askers and hinder the progress of knowledge sharing. Many prior works have investigated the issue of question quality on Stack Overflow [1, 5, 14]. Correa and Sureka [5] investigated closed questions on Stack Overflow, which suggests that a good question should contain enough code for others to reproduce the problem. Arora et al. [1] 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 [14] studied approaches to identifying unclear questions in CQA websites. However, all of these previous works focus on identifying the low-quality questions or how to increase the accuracy of the prediction, more in-depth research of improving the low-quality questions is still needed.

Based on our previous work [7], we present a web-based tool, named Code2Que, that helps developers post higher-quality questions on Stack Overflow. Developers can copy and paste their code snippets into our web application, then Code2Que helps developers to improve their posts by generating question titles as well as recommending related questions. The input to Code2Que is a code snippet, which is regarded as an ordered sequence of code tokens by our tool. Code2Que consists of an offline learning phase and an online recommendation phase. The output of Code2Que is two parts: (i) Generated Questions: Code2Que will generate a high-quality question title for a given code snippet based on our deep sequence-to-sequence model. Developers can utilize the generated questions for reformulating their posts. (ii) Retrieved Questions: Code2Que first embeds the code snippet into a vector, then searches through our codebase to retrieve relevant questions with similar problematic code snippets. No prior work has provided practical tools to identify related questions via measuring similarity between code snippets. To the best of our knowledge, our work is the very first to
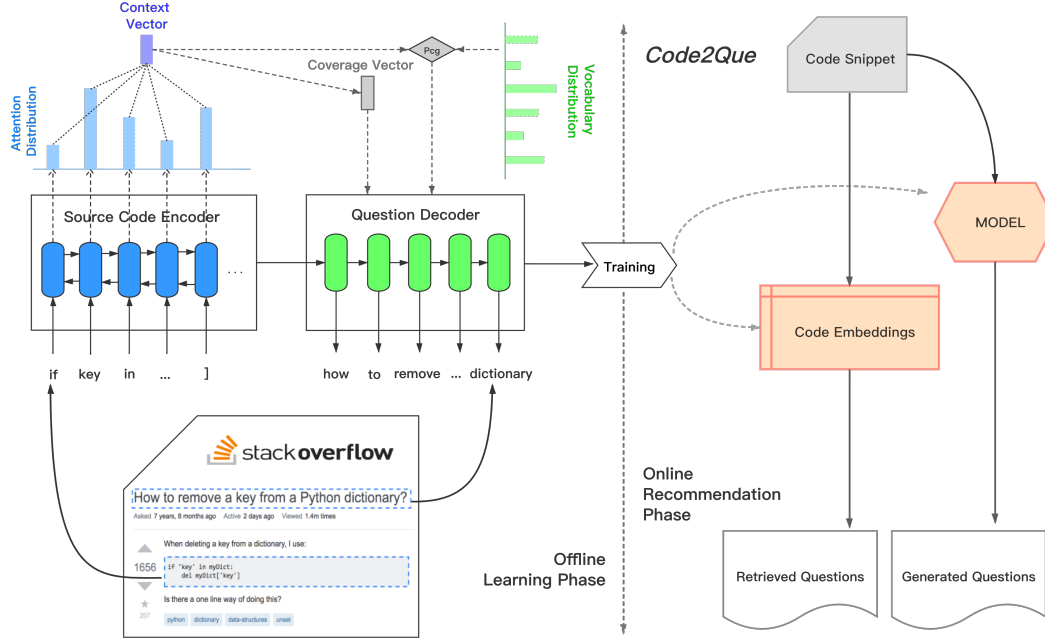
Figure 1: Overview of Our Approach

investigate the possibility of automatically improving low-quality questions in Stack Overflow.

CODE2QUE is able to benefit the following tasks: (i) Question Improvement. As many less experienced users lack the knowledge and terminology related to their problems or they have poor English writing skills, it is very hard, if not impossible, for them to always create clear and informative question titles. With the help of CODE2QUE, developers can use the questions generated by our approach to reformulate their low-quality questions. Developers can also quickly review a list of related questions in Stack Overflow of similar problematic code snippets. They thus can gain a better understanding of the problem and revise their earlier poorly asked questions. (ii) Code Embeddings. Our approach can embed code snippets into a fixed dimensional vector space, and with this a variety of applications such as code search (e.g. [4, 9]), clone detection (e.g. [6, 13]) and code summarization (e.g. [10, 11]), can benefit from the code embeddings used in our study.

The rest of the paper is organized as follows. Section 2 presents the overall workflow of our CODE2QUE approach and provides details of each step. Section 3 introduces the implementation details of CODE2QUE and its key usage scenarios. Section 4 shows the experimental results of our evaluation. Section 5 summarises our work.

2 APPROACH

2.1 Overview

Fig. 1 illustrates the overall framework of CODE2QUE. For a given code snippet, CODE2QUE assists developers in writing high-quality questions by automatically generating question titles and retrieving

the related questions in Stack Overflow. Our model contains two phases: offline learning and online recommendation.

In the offline learning phase, we first collect ⟨code snippet, question⟩ pairs from Stack Overflow posts. Since our goal is generating high-quality questions to help developers, we remove all the pairs in which the question score is less than 1. We train a deep sequence-to-sequence (seq2seq) model to map a code snippet directly to a high quality question title. Our offline learning model is divided into two components: a **Source-code Encoder** and a **Question Decoder**. The source code snippet is transformed by Source-code Encoder into a vector representation, with the help of an *attention* mechanism [2] to perform better content selection, a *copy* mechanism [8] to handle the rare-words problem, as well as a *coverage* mechanism [15] to avoid meaningless repetitions. The vector representation of the code snippet is then read by a Question Decoder to generate the target question titles.

In the online recommendation phase, for a given code snippet, the recommendation output is a Generated Question and a set of Retrieved Questions. The question title generated by the offline learning model can assist developers in writing high-quality questions that are more informative and clear. The retrieved similar questions can be used by developers to help them better understand their problems.

2.2 Offline Learning

2.2.1 Source-code Encoder. Our Source-code Encoder is a two-layer bidirectional LSTM network. Tokens in the code snippet are fed sequentially into the Source-code Encoder, which generates a sequence of hidden states. For example, given x_t is the input source code token at time step t, the Source-code Encoder will produce

the hidden states $\overrightarrow{\mathbf{h}}_t$ and $\overleftarrow{\mathbf{h}}_t$ 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.

2.2.2 Question Decoder. Our Question Decoder is a single-layer LSTM network, initialized with the state s produced by the Source-code Encoder. During training, at each time step t, the Question Decoder takes as input the embedding vector y_{t-1} of the previous word 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. The Question Decoder produces one symbol at a time and stops when the END symbol is emitted. The only change with the Question Decoder at testing time is that it uses output from the previous word, since there is no access to a ground truth then.

2.2.3 Incorporating the Attention Mechanism. One challenging task with the sequence-to-sequence model is dealing with the long sequence input. A solution was proposed by Bahdanau et al. [2], in which they introduced a technique called "*attention*" which significantly improved the performance of sequence-to-sequence models in machine translation systems. We incorporate the *attention* mechanism in our work and model the attention [2] distribution over words in the source code snippets, which allows the model to focus on the most relevant parts of the input sequence as needed.

2.2.4 Incorporating a Coverage Mechanism. Repetition is another challenge for attentional sequence-to-sequence models, where meaninglessly repeated words can be generated during the decoding process. As shown in Figure 2, "post" has been repeated twice by the attention model (highlighted with yellow color in Figure 2). To address this problem, we incorporate a *coverage* mechanism [15] to avoid meaningless repetitions. The *coverage* mechanism quantitatively emphasizes the coverage of sentence words and thus avoids generating repetitive text while decoding.

2.2.5 Incorporating a Copy Mechanism. Generating question titles from code snippets is a non-trivial task because the code snippets usually contain tokens with very rare occurrences, such as the word `get_client_ip` (highlighted with a blue color) in Figure 2. It is very difficult, often impossible, for a decoder to generate such a word solely based on language modeling. To address this challenge, we incorporate a *copy* mechanism [8] which allows the model to copy tokens from the source code snippet to the target generated question title. To do this, we maintain a binary classifier p_{cg} to determine whether to generate a word from the vocabulary or to copy the word directly from the input code snippet based on attention distribution. As shown in the last row in Figure 2, with the help of the *copy* mechanism, the mohdel properly picks up the method name `get_client_ip` from the source code snippet and copies it into the generated question titles.

2.3 Online Recommendation

2.3.1 Generated Questions. Once the offline learning model is trained, we do inference using a beam search on the pre-trained model. For a given code snippet, the Source-code Encoder encodes it into a fixed-dimensional real-valued vector, then the Question

Figure 2: Example of Generated Questions

Decoder reads the code embedding to infer the target question titles. The inference process stops when the model generates the END token which stands for the end of the sentence.

2.3.2 Retrieved Questions. To help developers better understand their problems, we retrieve other relevant questions in Stack Overflow according to the code snippet. After the offline training phase, each code snippet s_i in the training corpus is mapped to a fixed-dimensional vector c_i of real values. By stacking all the individual vectors together, we construct a source code snippet embedding matrix $C^{s \times d}$, where the first dimension s is the total number of code snippets and the second dimension d is the number of hidden states of the Source-code Encoder. After the developer submits his/her code snippet to our model, the code snippet is embedded into a vector by the Source-code Encoder, then we search through the code embedding matrix $C^{s \times d}$ to retrieve relevant questions with similar code snippets.

3 IMPLEMENTATION DETAILS

We have implemented CODE2QUE as a standalone web-based tool to assist developers in improving question titles in Stack Overflow. The source code and data can be found in our Github repository[1].

Data Collection. The data source of CODE2QUE is from the Stack Overflow data dump of September 2019[2]. We use the *Python*, *Java*, *Javascript*, *C#* and *SQL* tags to collect questions associated with the corresponding programming language. We extract the code snippet from each post's body by using ⟨*code*⟩ tags, and then pair it with its question title if the question score is higher than 1. We have collected more than 1 million ⟨*code snippet, question*⟩ pairs from Stack Overflow for different programming languages.

Data Preprocessing. We preprocess our collected data according to the following steps: We first tokenize the code snippet and question title using the NLTK toolkit [3] (Step1). After that, to avoid being overly context-specific, we use regular expressions to replace numbers and strings with special tokens "NUMBER" and "STRING" in code snippets (Step2). For question titles, we only keep the pairs if one of the interrogative keywords (e.g., "how", "what", "why", "which") appears in the question (Step3). Following that, we remove pairs where the code snippet and the question title are too long or

---

[1]https://github.com/beyondacm/Code2Que
[2]https://archive.org/details/stackexchange

Figure 3: Homepage of Code2Que

Table 1: Human Evaluation (Code2Que vs. Human)

Python / Java	Win (%)	Lose (%)	Tie (%)
Clearness	52.4 / 42.8	33.2 / 34.0	14.4 / 23.2
Fitness	55.2 / 47.2	24.0 / 39.6	20.8 / 13.2
Willingness	61.2 / 49.2	31.6 / 26.8	7.2 / 24.0

too short. We set the token range from 16 tokens to 128 tokens for code snippets and from 4 tokens to 16 tokens for question titles respectively. The remaining ⟨*code snippet*, *question*⟩ pairs are added to our training corpus.

Backend Model. We use the framework OpenNMT [12] to train our backend Model. We followed our previous experiment settings [7] for the training process. The collected ⟨*question*, *code snippet*⟩ pairs are input into the workflow of our approach described in Section 2, and the output consists of the trained sequence-to-sequence model and the associated code embeddings which are used by Code2Que as the backend model for online recommendation.

Frontend User Interface. Fig. 3 shows the home page of Code2Que. Code2Que provides an input box for developers to submit their source code snippet. After a developer submits his/her code snippet to the server, the code snippet is preprocessed and passed to our backend model. The outputs are organised into two separate result boxes for the generated question and retrieved questions respectively. For the generated question, Code2Que generates a question title according to the code snippet. For retrieved questions, Code2Que returns the top-5 most similar questions in our code base together with the links to their Stack Overflow posts.

System Optimization. To meet the efficiency requirement as an online web tool, we optimize the implementations of Code2Que: (i) Considering that the offline learning model and code embeddings are frequently used in the online recommendation phase, we put them into cache to reduce redundant data loading. (ii) We created an index for the relevant questions in our database to speed up retrieval. (iii) We used locality sensitive hashing for fast nearest neighbour search in our large code embeddings data sets.

User Scenarios. Fig. 3 demonstrates an example[3] of using our tool to improve the low quality question titles in Stack Overflow. The original question title, i.e., "*Python [] notation in list*", was unclear and uninformative. When developer submits the code snippet to our tool, the generated question title is "how to add an item to a

---

[3]https://stackoverflow.com/questions/12828968/python-notation-in-list

list". The newly generated question title is much more clear and also questioning on the key problems of the user's concern. After going through the generated question titles and relevant questions retrieved by our approach, developers can gain a better understanding of the key problem. As a result, developer can use our tool to reformulate their earlier poorly asked questions and write questions more effectively.

4 EVALUATION AND USER STUDY

To evaluate whether Code2Que can generate better question titles for low-quality questions in Stack Overflow, we performed a user study on our Python and Java datasets. We sampled 50 low-quality ⟨*question*, *code snippet*⟩ pairs, which have been marked as lacking clarity and/or need to be further improved upon. For each code snippet, we conducted a pairwise comparison between two question titles, the original title written by humans and the title generated by our Code2Que. For each pairwise comparison, we asked 5 evaluators to decide which one is better in terms of three metrics: *Clearness*, *Fitness*, and *Willingness*; tie was allowed. *Clearness* measures whether a question title is expressed in a clear way. *Fitness* measures whether a question title is reasonable in logic with the provided code snippet. *Willingness* measures whether a user is willing to respond to a specific question.

Evaluation results are summarized in Table 1. We can see that: (i) question titles generated by Code2Que outperform the low-quality question titles for all metrics. This demonstrates that our approach is able to produce clearer and/or more appropriate questions, indicating the ability of Code2Que to improve low-quality Stack Overflow questions. (ii) Code2Que question titles have better willingness scores. This shows that our question titles are more likely to elicit further interactions, and helpful to increase the likelihood of receiving answers. (iii) Not all of the poor quality question titles can be improved by our approach. Even though our question titles are not perfect, our Code2Que is the first work on this direction. We release our tool and dataset to inspire further follow-up research.

5 SUMMARY AND FUTURE WORK

We demonstrate Code2Que, a web-based tool for improving low-quality question titles in Stack Overflow. Developers copy and paste their code snippets into our web application. Code2Que automatically generates question titles for the code snippets via a deep sequence-to-sequence model. Developers can utilize the generated questions for their posts. Code2Que also searches for related questions with similar problematic code snippets to help developers better understand their problems. Human evaluation results show that for a large number of low-quality questions in Stack Overflow, Code2Que can improve the quality of the question titles in terms of *Clearness*, *Fitness*, and *Willingness*. In future work, we will design better models to generate question titles by considering more context information.

ACKNOWLEDGMENT

REFERENCES

[1] 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.

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

[3] 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.

[4] Jose Cambronero, Hongyu Li, Seohyun Kim, Koushik Sen, and Satish Chandra. 2019. When deep learning met code search. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 964–974.

[5] 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.

[6] Yi Gao, Zan Wang, Shuang Liu, Lin Yang, Wei Sang, and Yuanfang Cai. [n.d.]. TECCD: A Tree Embedding Approach for Code Clone Detection. In *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 145–156.

[7] Zhipeng Gao, Xin Xia, John Grundy, David Lo, and Yuan-Fang Li. 2020. Generating Question Titles for Stack Overflow from Mined Code Snippets. *arXiv preprint arXiv:2005.10157* (2020).

[8] 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).

[9] 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.

[10] 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.

[11] 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.

[12] Guillaume Klein, Yoon Kim, Yuntian Deng, Jean Senellart, and Alexander M. Rush. 2017. OpenNMT: Open-Source Toolkit for Neural Machine Translation. In *Proc. ACL*. https://doi.org/10.18653/v1/P17-4012

[13] Liuqing Li, He Feng, Wenjie Zhuang, Na Meng, and Barbara Ryder. 2017. Cclearner: A deep learning-based clone detection approach. In *2017 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 249–260.

[14] Jan Trienes and Krisztian Balog. 2019. Identifying Unclear Questions in Community Question Answering Websites. In *ECIR*.

[15] Zhaopeng Tu, Zhengdong Lu, Yang Liu, Xiaohua Liu, and Hang Li. 2016. Modeling coverage for neural machine translation. *arXiv preprint arXiv:1601.04811* (2016).