Improving Code Summarization Through Automated Quality Assurance

Yuxing Hu[1,2], Meng Yan[2,3*], Zhongxin Liu[4], Qiuyuan Chen[4], Bei Wang[1,2]
[1]Key Laboratory of Dependable Service Computing in Cyber Physical Society (Chongqing University), Ministry of Education, China
[2]School of Big Data and Software Engineering, Chongqing University, Chongqing, China
[3]Pengcheng Laboratory, Shenzhen, China
[4]College of Computer Science and Technology, Zhejiang University, Hangzhou, China
Email:{yuxhu, mengy, bwang2013}@cqu.edu.cn,{liuzx, chenqiuyuan}@zju.edu.cn

2021 IEEE 32nd International Symposium on Software Reliability Engineering (ISSRE) | 978-1-6654-2587-2/21/$31.00 ©2021 IEEE | DOI: 10.1109/ISSRE52982.2021.00057

***Abstract*—The code summarization task aims to generate brief descriptions of source code automatically. It is beneficial for developers to understand source code. However, almost all of current code summarization approaches may generate low-quality (BLEU4<40) summaries, which will mislead developers. Previous work has shown that it is possible to conduct quality assurance for document generation (QA4DG) and improve the practicability of document generation approaches. Code summarization can also be regarded as a document generation task. This work aims to investigate whether QA4DG approaches can be leveraged to improve code summarization.**

Specifically, we first investigate whether existing QA4DG approaches can be plugged in code summarization approaches. We find that an automated quality assurance framework for commit message generation named QACom performs best. Inspired by the idea behind QAcom, we propose an ensemble code summarization approach called Ensum. Precisely, given a code snippet, Ensum first uses current code summarization approaches to generate candidate summaries. Then, Ensum predicts the quality of each candidate summary using a collaborative filtering-based component and a retrieval-based component and selects the best candidate summary as the output. Experimental results on two public datasets show that Ensum outperforms three state-of-the-art single approaches and one ensemble approach for code summarization in terms of BLEU-4, METEOR, and ROUGE-L.

***Index Terms*—Code Summarization, Code Comment, Quality Assurance**

I. Introduction

A code summary is a natural language description of a code snippet that helps the developer understand the meaning of this code without reading the entire source code. Considering that developers often spend a lot of time on source code comprehension [1], high-quality code summaries are essential for software development and maintenance. However, writing code summaries manually is tedious and time-consuming, which increases the need for automatic code summarization approaches.

Many code summarization approaches have been proposed [2]–[7]. Recently, with the advances of deep learning and the large and ever-increasing amount of source code data, leveraging deep learning models to automatically learn to generate code summaries from massive code-summary pairs has become a very popular research topic. For example, Hu et al. [2] propose a novel neural network model which combines both the lexical and the structural information of code snippets to generate code summaries. Zhang et al. [4] propose a retrieval-based code summarization approach called Rencos, which enhances neural network model with the most similar code snippets retrieved from the training set. We refer to this topic as *"neural code summarization"*.

```
public static void finish_activity_on_phone ( context context )
{
        if ( context instance_of_enhancedbookmark activity )
                {
                        ( ( activity ) context ) . finish ( ) ;
                }
}
```

reference: closes the enhancedbookmark activity on phone .

Hybird-Deepcom: add a new location .

NMT: helper method to finish the activity .

Rencos: called when the user has clicked the battle .

Fig. 1: An example where the summaries generated by Deepcom, NMT, and Rencos are all low-quality summaries.

Although existing neural code summarization approaches are shown to have promising performance and can generate many high-quality code summaries, according to previous studies, some code summaries generated by existing code summarization approaches obtain BLEU-4 scores less than 40, which are considered to be low-quality summaries [8]. Figure 1 shows an example where the code summaries generated by three state-of-the-art code summarization approaches are of low quality. These summaries may mislead developers while taking extra developer time to filter.

In fact, almost all neural document generation approaches suffer from the problem mentioned above [9], [10]. To alleviate this problem, researchers have proposed several quality assurance approaches [11], [12] for document generation tasks like neural machine translation (NMT) [11] and commit message generation (CMG) [12]. Such approaches aim to automatically predict the quality of generated documents and filter out the ones that are predicted to be of low quality, and can improve the practicability of NMT and CMG approaches. We refer

*Corresponding author.

2332-6549/21/$31.00 ©2021 IEEE
DOI 10.1109/ISSRE52982.2021.00057

to this kind of idea as QA4DG. However, no prior work investigates whether QA4DG approaches can be applied to improve code summarization.

To bridge this gap, in this work, firstly we conduct an empirical study to investigate whether existing QA4DG approaches can be applied to code summarization to filter out generated low-quality code summaries and improve the practicability of code summarization approaches. Specifically, we apply three QA4DG approaches, i.e., collaborative filtering (CF) [13], QAcom [11], and Test-NMT [12] to three popular code summarization approaches, namely Deepcom [2], Rencos [4], and NMT [7]. Given a QA4DG approach and a code summarization approach, the QA4DG approach predicts the quality of each summary generated by the code summarization approach and filters out the predicted low-quality ones. Four metrics, i.e., BLEU-4 [14], METEOR [15], ROUGE-L [16] and Preserved-Ratio [11], are calculated on the preserved generated summaries to measure the effectiveness of the QA4DG approach. Experiments show that QAcom [11] can achieve a BLEU-4 close to 40 with the highest Preserved-Ratio. Therefore, we believe that QAcom is the most applicable approach for code summarization.

Secondly, inspired by the idea behind QA4DG approaches, i.e., automatically predicting the quality of generated documents, we propose an **en**semble code **sum**marization approach called Ensum. The core idea of Ensum is that given a code snippet and multiple code summarization approaches, we can automatically predict the quality of the summaries generated by the code summarization approaches and choose the one predicted to be the best as the final generated summary. Specifically, based on the finding of our empirical study, Ensum adopts the technique of QAcom to perform summary quality prediction, i.e., predicts the quality of a summary based on a collaborative filtering-based component and a retrieval-based component. It integrates three code summarization approaches, namely NMT, Rencos and Hybrid-Deepcom (hereon, Deepcom), all of which are popular state-of-the-art approaches. We evaluate the effectiveness of Ensum on two public datasets in terms of BLEU, METEOR, and ROUGE-L, and compare them with the three single-model approaches and a state-of-the-art ensemble approach named Codesum [17]. It is worth mentioning that Codesum is a supervised approach requiring manual efforts to label code comment types, while Ensum is unsupervised and does not require human-labeled information. Experimental results show that Ensum can well integrate multiple single-model code summarization approaches by effectively identifying the generated code summaries with the best quality, and the unsupervised Ensum also outperforms the supervised Codesum in terms of all metrics.

In summary, the contributions of this paper include :

- We conduct an empirical study to investigate whether QA4DG approaches can be applied to code summarization to filter out low-quality generated code summaries. We plug three QA4DG approaches in three popular code summarization approaches, respectively, and find QAcom is the best performed QA4DG approach. This paper is the first study to explore the impact of existing QA4DG approaches on code summarization.
- We propose an ensemble code summarization approach Ensum, which can effectively integrate multiple single-model code summarization approaches by predicting the quality of their generated summaries.
- We leverage Ensum to integrate three single-model code summarization approaches and conduct comprehensive experiments to evaluate Ensum on two datasets. The results confirm that Ensum is effective and outperforms its used single-model approaches and a state-of-the-art ensemble approach.
- We open source our replication package, including the dataset and the source code of our study [18].

The rest of the article is organized as follows: Section II introduces the background. Section III describes the experimental setup. Section IV details the empirical study on applying QA4DG approaches to code summarization. Section V proposes Ensum and describes the evaluation methods and results of Ensum. Section VI discusses the threats to validity. Section VII reviews related studies. Section VIII concludes this paper.

II. Background

In this section, we briefly introduce the background of code summarization and quality assurance for document generation (hereon, QA4DG).

A. Code Summarization

Code summarization task aims to generate a brief description for a given code snippet. In recent years, with the advances of deep learning and the large and ever-increasing amount of source code data, leveraging neural network to perform automatic code summarization has attracted a lot of attention and become a hot research topic [2], [4], [6], [7], [19]. We refer to it as neural code summarization.

The basic idea of neural code summarization is that we can leverage neural networks to learn the transition between code and summaries from massive code-summary pairs collected from existing software projects. Those neural code summarization approaches generally consist of two parts, i.e., the encoder part and the decoder part. The encoder part is composed of one or more neural components, namely encoders, to capture different kinds of information, i.e., lexical information and structure information, of the given code snippet and encode such information into real value vectors. Such vectors can be regarded as the intermediate representation of the input code snippet, and will be input into the decoder part. The decoder part usually contains only one neural component called decoder, and is responsible for generating summary based on the output of the encoder part. RNN and its variants are commonly-used decoders. Decoders are used to model the conditional probability of generating a summary token conditioned on the input code snippet and the summary tokens that have already been generated. Recently, some approaches

[4] also integrate retrieval methods into neural networks to enhance the performance of code summarization.

B. Quality Assurance for Document Generation

The idea of quality assurance for document generation (QA4DG) is to predict the quality of the documents generated by document generation approaches, and filter out the ones that are predicted to be of low quality before presenting generated documents to readers. In this way, we can improve the overall quality of the generated documents that are presented to readers and consequently improve the practicability of document generation approaches. Existing QA4DG approaches generally calculate quality scores based on the generated documents, the input from which the documents are generated and the training set to measure the quality of generated documents. They differ from each other in how they compute quality scores. Currently, researchers have proposed QA4DG approaches for document generation tasks such as neural machine translation [12] (NMT) and commit message generation [11], [20] (CMG). Unfortunately, whether QA4DG approaches can be used to improve code summarization is still uninvestigated.

III. Experiment Setup

In this work, we first conduct an empirical study to investigate whether quality assurance for document generation approaches can be applied to code summarization, and then propose and evaluate an ensemble code summarization approach called Ensum that is inspired by the idea of QA4DG. This section presents the datasets, the selected QA4DG approaches and code summarization approaches and the evaluation metrics code summarization used in our empirical study and the evaluation of Ensum.

A. Datasets

We use the dataset provided by Hu et al. [2], which is collected from 9,714 GitHub projects. This dataset consists of 588,108 code-summary pairs. For this dataset, we used its two settings:

Within-project dataset: It does not distinguish between projects, with a training set consisting of 445,812 code-summary pairs and a validation set and a test set of 20,000 pairs each.

Cross-project dataset: Its training set consists of 455,000 code-summary pairs, the validation set and the test set contain 15,606 pairs each. The validation and the test sets do not overlap with the training set in terms of the projects where the pairs are collected from.

Table I and Table II present the statistics of the two datasets, including the number of code tokens, code lines, summary words, and the mean, median, minimum, and quantile length of codes and summaries. It can be seen that the data distributions of the training set, validation set, and test set are consistent on both datasets.

B. Code Summarization Approach Selection

We select three state-of-the-art approaches from the existing neural code summarization approaches, i.e., Hybrid-DeepCom and Rencos. When using these approaches, we follow their original settings.

NMT [7]: Neural machine translation model. Here we use a RNN with global attention [21], which is a baseline commonly used in the code summarization domain. We input code tokens to the encoder. And summaries output from the decoder. We implement the model using the OpenNMT [7] and adopt the default settings.

Hybrid-DeepCom [2]: Hybrid-Deepcom, for short Deepcom, exploits a deep neural network that combines the lexical and syntactic information of Java methods for better comments generation. Specifically, the lexical information learns from source code, and the code will be processed as tokens. The syntactic information learns from abstract syntax tree (AST) sequences which uses javalang [22] to process. We input code tokens and AST sequences to encoder and Deepcom outputs summaries from decoder.

Rencos [4]: Rencos is the first that tries to enhance neural code summarization approaches with retrieval. It consists of a retrieval-based component and a neural-based component. For the retrieval-based component, it calculates the similarity of the new code segment to the historical code to find the most similar one. We input the code tokens as context vectors through encoder and retrieval component. The decoder then computes the conditional probabilities and generates the final summaries.

C. Quality Assurance Approach Selection

We select three state-of-the-art QA4DG approaches to evaluate their effectiveness on code summaries. It is important to note that for all three QA4DG approaches they construct an objective function and then train it on the validation set using a differential evolutionary algorithm [23].

Collaborative Filtering (CF) [13]: This approach uses a collaborative item-based filtering algorithm to calculate two separate scores for both under-translated and over-translated cases. Under-translated means some essential words or phrases are missing in the generated documents. Over-translated means some words or phrases in the generated documents are unnecessary.

Specifically, CF uses an item-based collaborative filtering algorithm to discover the similarity between items and items, based on all users' ratings of the items, then recommends similar item information to that user based on the user's historical preference information. For the code summarization task, CF considers a function as a user and each word in code or summary as an item. And then CF can construct a mapping of associated words in summaries for each word in code that corresponds to it. Based on these mappings, CF can calculate precision and recall for under-translation and over-translation respectively. Thus CF can get two quality scores. If they are below the thresholds, CF will treat this as a low-quality summary and filter it.

TABLE I: Count of the Datasets

Type	Within-project dataset				Cross-project dataset			
	Pairs	Code Tokens	Code Lines	Summary Words	Pairs	Code Tokens	Code Lines	Summary Words
Train	445,812	24,889,887	1,834,662	4,571,644	455,000	20,515,239	1,873,214	4,671,191
Valid	20,000	1,112,533	82,207	204,357	15,406	699,983	62,526	155,539
Test	20,000	1,103,447	81,573	205,476	15,406	698,458	62,701	154,747

TABLE II: Statistics of the Datasets

Type	Within-project dataset							Cross-project dataset						
	Mean	Std.	Min.	1st Quart	Med.	3rd Quart.	Max.	Mean	Std.	Min.	1st Quart	Med.	3rd Quart.	Max.
Train-Code	55.83	53.07	5	18	36	75	665	45.44	42.60	5	14	29	63	199
Train-NL	10.25	4.48	1	7	9	13	32	10.09	4.36	1	7	9	12	30
Valid-Code	55.63	52.41	5	18	36	75	397	45.08	42.56	5	15	28	61	199
Valid-NL	10.22	4.46	1	7	9	13	30	10.27	4.49	1	7	9	13	32
Test-Code	55.17	52.54	5	18	35	74	365	45.33	42.57	5	14	29	62	199
Test-NL	10.27	4.49	2	7	9	13	29	10.05	4.36	1	7	9	12	29

Test-NMT [12]: Test-NMT addresses the current problem of faulty NMT translations and proposes an approach to check the translated content. It also divides the problem into two parts: under-translated content, over-translated content.

For under-translated, a collaborative filtering algorithm is used. Thus it can calculate a quality score for code summarization according to CF we just mentioned. For over-translated, the algorithm is based on the frequency of words appearing in the translation. Precisely, the algorithm will count the number of times each word appears in the translation. If it occurs more than once, it is considered to be over-translated. The over-translated words will be removed and not to calculate quality score. Thus Test-NMT will obtain one quality score. If it is below the threshold, Test-NMT will treat this as a low-quality summary and filter it.

Quality Assurance for Commit Message (QAcom) [11]: This approach is used to automatically ensure the quality of generated messages and consists of a collaborative filtering-based component and a retrieval-based component. Each component will produce a quality score to measure the quality of a generated document.

The collaborative filter-based component has two cases of over-translation and under-translation [12]. Therefore, the CF component will calculate the collaborative filtering score (CF score) by checking whether each generated summary is under-translated or over-translated for code summarization.

The retrieval-based component uses the similarity between the current code and the historical code to calculate the retrieval score for code summarization. And it will use BLEU-4 to represent the similarity so retrieval score can be calculated. Thus QAcom will get two quality scores from the two components. If they are below the thresholds, QAcom will treat this as a low-quality summary and then filter it.

D. Evaluation Metrics

We use several evaluation metrics commonly used in code summarization and QA4DG, BLEU-4 [14], Meteor [15], ROUGE-L [16], Preserved-Ratio [11], Precision, and Recall.

BLEU-4 measures the n-gram precision between X and Y by computing the overlap ratios of n-grams and applying brevity penalty on short translation hypotheses. Given the generated summary X and the corresponding references Y, $BLEU-N$ calculates the similarity by computing the n-gram precision of the candidate sentences for the reference sentences and penalizes the lengths that are too short. BLEU-1/2/3/4 correspond to 1-gram, 2-gram, 3-gram, and 4-gram scores, respectively. The formula to calculate BLEU-N (N = 1, 2, 3, 4) is:

$$BLEU-N = BP * exp\left(\sum_{n=1}^{N} w_n logp_n\right) \tag{1}$$

where p_n is the precision score of n-gram matching between candidate and reference sentences, b_p is the brevity penalty score, and w_n is the uniform weight $\frac{1}{N}$. BP stands for Brevity Penalty and its formula is

$$BP = \begin{cases} 1 & ,if\ cand < ref \\ e^{1-\frac{ref}{cand}} & ,if\ cand \geq ref \end{cases} \tag{2}$$

where $cand$ and ref represent the length of the candidate and reference sentences, respectively. In particular, in this paper we use BLEU-4 because it is widely used by code summarization approaches for evaluation.

METEOR is based on word matches between the generated summaries and its reference. It is calculated using the F-score of word matches and a fragmentation penalty for considering word order differences. METEOR calculates sentence-level similarity scores by aligning the generated summaries to the corresponding references with the formula:

$$Meteor = (1 - \gamma \cdot frag^{\beta}) \cdot \frac{P \cdot R}{\alpha \cdot P + (1-\alpha) \cdot R'} \tag{3}$$

where P and R are single word precision and recall, and $frag$ is the fragmentation fraction. γ, β and α are three penalty parameters with default values of 0.9, 3.0 and 0.5, respectively.

ROUGE-L is widely used in text summarization and provides F-score based on longest common subsequence. ROUGE-L provides F scores based on the longest common subsequence (LCS). Suppose the lengths of X and Y are m and n, ROUGE-L is computed as follows:

$$P_{LCS} = \frac{LCS(X,Y)}{m} \tag{4}$$

$$R_{LCS} = \frac{LCS(X,Y)}{n} \tag{5}$$

$$F_{LCS} = \frac{\left(1+\beta^2\right) R_{LCS} P_{LCS}}{R_{LCS} + \beta^2 P_{LCS}} \quad (6)$$

where $\beta = P_{LCS}/R_{LCS}$ and F_{LCS} is the value of ROUGE-L.

Preserved-Ratio measures the retention of high quality summaries by QA4DG approaches. It calculates the percentage of preserved summaries to all summaries with the formula:

$$Preserve = \frac{summary_{pre}}{summary_{all}} \quad (7)$$

where $Preserve$ is the preserved percentage, $summary_{pre}$ is the number of preserved summaries, and $summary_{all}$ is the number of all summaries.

We also use **Precision** and **Recall** to evaluate the effectiveness of QA4DG approaches.

$$Precision = \frac{TP}{TP + FP} \quad (8)$$

$$Recall = \frac{TP}{TP + FN} \quad (9)$$

where True Positive (TP) represents the number of the comments that are correctly predicted to high quality; False Positive (FP) represents the number of the comments that are wrongly predicted to high quality; False Negative (FN) represents the number of the comments that are wrongly predicted to low quality.

IV. Quality Assurance for Code Summarization

In this section, we introduce the overview and empirical study of quality assurance for code summarization.

A. The Overview of Our Empirical Study

Figure 2 shows the overview of quality assurance for code summarization. The quality assurance for code summarization involves the following four steps: data preprocessing, using code summarization approaches to generate code summaries, applying quality assurance approaches to code summaries, and filtering low-quality code summaries.

First, we preprocess the dataset for each code summarization approach following the original data preprocessing procedure of this approach. The neural model of each code summarization approach is trained on the training set. During the model training phase, different code summarization approaches can be used. We select three popular code summarization approaches, i.e., NMT, Deepcom and Rencos. The model will then generate code summaries for each code snippet in the test set. Subsequently, we apply three QA4DG approaches namely CF, TestNMT and QAcom to filter out the low-quality summaries.

Specifically, for three QA4DG approaches we will construct an objective function and then train it on the validation set using a differential evolutionary algorithm. To find the optimal thresholds of quality scores, we set the objective function to preserve as many generated summaries as possible while ensuring that the BLEU scores of the preserved summaries exceed 40. The BLEU scores above 40 are based on the fact that we consider a set of generated summaries to be of high quality if their BLEU scores exceed 40 [8]. After finding the optimal thresholds, each QA4DG approach will calculate the quality scores. If the quality scores are below the thresholds, QA4DG will treat this as a low quality summary and filter it.

B. Empirical Study on Code Summarization

We conduct an empirical study to investigate whether existing QA4DG approaches can be applied to code summarization to filter out generated low-quality code summaries and improve the practicability of code summarization approaches.

Specifically, we use BLEU-4, METEOR, and ROUGE-L to automatically evaluate these approaches. For example, given a dataset and a code summarization approach, we will first evaluate the summaries generated by this approach on the test set. Next, we use the three quality assurance approaches to filter out the summaries generated by this code summarization approach for quality assurance. Subsequently, we can obtain three different filtering results from each of the three quality assurance approaches. In addition, we will calculate the ratio of preserved summaries to all summaries, i.e., the Preserved-Ratio. Finally, by comparing the filtering results, we choose the quality assurance approach whose result is close to or above 40 on BLEU-4, and preserving as many summaries as possible.

Once we have selected the most effective quality approach for code summarization by automatic evaluation, we need to validate the approach's performance further. Therefore, we also perform the human evaluation. Based on a previous study [9], [20], we invite four master students in software engineering to participate, all of whom have four years of java programming experience. We randomly select 100 functions and the corresponding summaries by the three code summarization approaches from the two datasets. The four participants need to compare the reference with the generated summaries and assign a score to each generated summary. Scores ranged from 1 to 5. A score of 1 indicates no association between the two summaries, and a score of 5 tells that the two summaries are identical or have the same meaning. In particular, we consider summaries with a score of 4 or 5 to be of high quality and the rest of the messages to be of low quality. The scoring range and criteria are consistent with Liu et al. [9]. For each function, we obtain scores from four participants. The final score of the function is obtained by averaging the four scores. Finally, we will compare the distribution of scores and Preserved-Ratio in the human evaluation results with the automatic assessment results.

C. Experimental Results

Automatic Evaluation. Table III shows the results of our experiments. The Rencos in Within-project dataset is used as an example. The "Rencos" row represents the performance of Rencos on the test set since the quality assurance approach has not been used on the dataset at this point, and therefore the Preserved-Ratio is 100%. The "Rencos+CF" row represents the overall quality of the preserved code summaries after applying CF to Rencos. We can observe from Table III that:

(1) All three existing quality assurance approaches improve code summarization. For example, on the Within-project

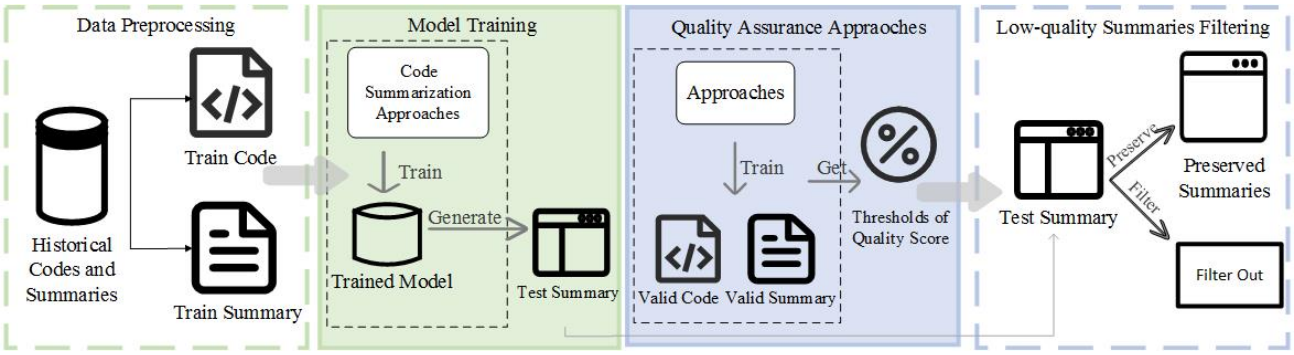

Fig. 2: Overview of Quality Assurance for code summarization.

TABLE III: Effectiveness of Three Quality Assurance Approaches on Two Datasets.

Approach	Within-project dataset				Cross-project dataset			
	Bleu-4	METEOR	ROUGE-L	Preserve-Ratio	Bleu-4	METEOR	ROUGE-L	Preserve-Ratio
Deepcom	0.326	0.249	0.513	100%	0.205	0.163	0.385	100%
Deepcom+CF	0.353	0.277	0.579	32.36%	**0.453**	**0.308**	**0.586**	35.78%
Deepcom+Test-NMT	0.355	0.264	0.501	63.07%	0.270	0.196	0.430	**71.74%**
Deepcom+QAcom	**0.482**	**0.337**	**0.617**	**68.27%**	0.397	0.273	0.541	42.91%
Rencos	0.281	0.210	0.458	100%	0.210	0.162	0.382	100%
Rencos+CF	0.383	**0.286**	**0.571**	44.22%	0.385	0.267	0.539	40.65%
Rencos+Test-NMT	**0.398**	0.280	0.529	41.45%	0.239	0.183	0.412	**80.00%**
Rencos+QAcom	0.376	0.268	0.526	**67.23%**	**0.477**	**0.317**	**0.598**	35.18%
NMT	0.255	0.197	0.457	100%	0.165	0.137	0.349	100%
NMT+CF	0.379	0.271	0.543	26.10%	**0.422**	0.290	**0.576**	22.87%
NMT+Test-NMT	**0.407**	**0.284**	**0.561**	37.54%	0.202	0.157	0.373	**70.26%**
NMT+QAcom	0.394	0.274	0.550	**60.10%**	0.404	**0.316**	0.541	31.02%

dataset, compared with Deepcom, Deepcom+CF goes up 8%, 10%, 11% on BLEU-4, METEOR, ROUGE-L, respectively. Deepcom+QAcom goes up 48%, 35%, 20%, respectively. Deepcom+Test-NMT goes up 9%, 6% on BLEU-4, METEOR, and no improvement on ROUGE-L, respectively. We can see that the existing QA4DG approaches can be applied to code summarization and enhance code summarization. In addition, all three approaches use a differential evolutionary algorithm to adjust the thresholds, but there are still some approaches that generate summaries with BLEU-4 below 40, which means these approaches cannot filter out low-quality summaries better for the current dataset.

(2) The overall quality of code summaries preserved by QAcom is the best. First, QAcom can raise BLEU-4 above 40 in all cases, while many QA4DG approaches cannot do so. Secondly, for those QA4DG approaches that can reach BLEU-4 above 40, QACom achieves a higher Preserved-Ratio than them. For example, on the dataset Cross-project dataset, NMT+QAcom can improve on average 145%, 131%, and 60% over NMT in terms of BLEU-4, METEOR, and ROUGE-L, respectively. In contrast, NMT+CF can only improve 33%, 27%, and 16%, respectively, while NMT+Test-NMT can improve 40%, 33%, and 19%, respectively. In addition, it can be seen on the Within-project dataset that the Preserved-Ratio of Deepcom+Test-NMT and Deepcom+QAcom are basically the same. However, Deepcom+QAcom is 36%, 28%, and 23% higher than Deepcom+Test-NMT on BLEU-4, METEOR, and ROUGE-L, respectively. Therefore, we can infer that QAcom is the most effective quality assurance approach for improving code summarization.

(3) The overall good results of QAcom are due to the combination of collaborative filtering and retrieval methods. For example, on the Within-project dataset, the Preserved-Ratio of Deepcom+Test-NMT is close to QAcom. However, it is lower than QAcom on other evaluation metrics, which means the quality of preserved summaries by Test-NMT is generally inferior to QAcom. Therefore, we believe that QAcom can better filter out low-quality summaries by combining collaborative filtering and retrieval methods and preserve as many high-quality summaries as possible.

(4) The BLEU scores of all three code summarization approaches that use QAcom are close to or exceed the BLEU constraint we use during training, i.e., 40, indicating that the overall quality of the code summaries preserved by QAcom is good. Thus, QAcom can guarantee the overall quality of the code summaries.

In addition, the Preserved-Ratio of QAcom is related to the used dataset and code summarization approach. For example, on the Within-project dataset, the Preserved-Ratios of all three approaches are higher than 60%. On the Cross-project dataset, Deepcom+QAcom, which has the highest Preserved-Ratio, preserved only 42.91% of the summaries. The reason may be that on the Cross-project dataset data, Deepcom is no longer able to generate better code summaries. Based on these observations, we believe that QAcom can effectively guarantee the quality of code summaries generated by code

summarization approaches and improve the practicability of existing code summarization approaches.

Human Evaluation. After we select the most effective approach QAcom, we also conduct the human evaluation of QAcom. Table IV and Table V show the results of the human evaluation. The "1" to "5" columns show the distribution of the final scores of each code summarization approach on each dataset. The "Preserved" column refers to the number of preserved summaries, and the "Mean-score" column represents the average score of the scores. We can observe Table IV that:

(1) The distribution of the scores is consistent with the results of the automatic evaluation shown in Table IV. For example, in the Within-project dataset, Deepcom has the highest percentage of scores of 2 and 3. While Deepcom+QAcom has the highest rate of 4 and 5. This indicates that the results filtered by QAcom have the highest percentage on 4 and 5 scores, meaning that QAcom can preserve high-quality summaries.

(2) "Preserved" is consistent with the Preserved-Ratio of the automatic evaluation shown in Table III. For example, Deepcom's Preserved-Ratio on the Within-project dataset is 68.27%, and Deepcom's "Preserved" is 68 out of 100. This indicates that QAcom does filter out a large number of low-quality code summaries. In all cases, QAcom can filter out more low-quality summaries and preserve more high-quality summaries.

(3) The average score of code summaries preserved by QAcom is higher than the average score of the original approach. For example, the average score of Deepcom on the Within-project dataset is 2.84, and the average score improves to 3.58 after QAcom filtering.

(4) As shown in Table V, we also calculate the precision and recall of QAcom about low-quality and high-quality summaries. We note that QAcom consistently outperforms random selection. Therefore, randomly preserving generated summaries hardly affects the overall quality.

Our human evaluation results confirm the ability of QAcom to guarantee the quality of the summaries generated by the code summarization approaches and further demonstrate the effectiveness of QAcom in filtering low-quality code summaries and preserving high-quality summaries.

Existing quality assurance approaches can improve code summarization, and QAcom can maximize code summaries and ensure the quality of the summaries while preserving as many high-quality summaries as possible.

V. Ensum: an Ensemble Code Summarization Approach

Based on the above empirical study, we also notice that the idea behind QA4DG approaches, i.e., predicting the quality of generated documents, can also be used to construct ensemble code summarization approach. Inspired by this, we propose an ensemble code summarization approach called Ensum. This section describes the motivation, the details, the evaluation, and the evaluation results of Ensum.

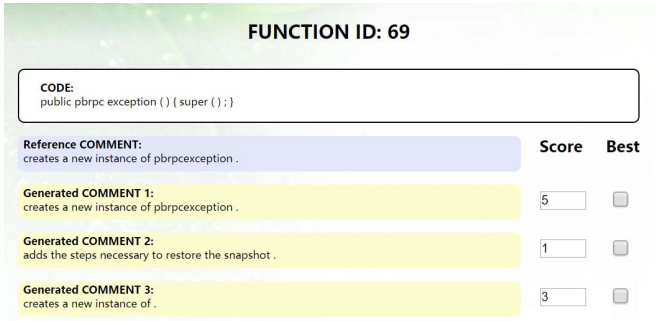

Fig. 3: An example of human evaluation scores of the summaries generated by different approaches showing that only one approach performs well.

A. Motivation of Ensum

As shown in Figures 3, during the human evaluation, we find that for the same function, only one of the summaries generated by the three code summarization approaches may perform well, i.e., get a score of 5. Therefore, we do an automatic complementarity analysis on the high-quality summaries preserved by the three code summarization approaches after filtering. As shown in Table VI, good(only) means that only the summary generated by the current approach is of high quality compared with its reference. Good(all) means that the summaries generated by all three approaches are of high quality the current reference. For example, on the Within-project dataset, Deepcom generates 14 unique high-quality summaries, Rencos generates 17 unique summaries, and NMT generates 8 unique summaries.

The above phenomena suggest that the three code summarization approaches are complementary to each other so we can take advantages of them. Therefore, we propose the Ensum approach to integrate the three selected complementarity approaches for improving complementarity. In addition, we have obtained through empirical research that the quality assurance approach that maximizes complementarity is QAcom. Therefore, we consider whether we can apply the idea of QAcom to Ensum, which enhances the quality of complementarity.

B. Overview of Ensum

Figure 4 shows the overview of the Ensum. On the one hand, we take advantage of the fact that the three code summarization approaches have complementary characteristics to integrate the three approaches. On the other hand, we use the idea of QAcom to filter out low-quality summaries to ensure the quality of code summarization. Thus, we can evaluate the summaries generated by the three summarization approaches using the idea of QAcom. Finally, for each code block, the best quality summary is kept as the final result.

Specifically, Ensum consists of a collaborative filtering-based component and a retrieval-based component. Both components will use the corresponding quality scores to evaluate the summaries generated by each of the three summarization

TABLE IV: The Human Evaluation Results of QAcom on two datasets.

Dataset	Approach	1	2	3	4	5	#Preserved	Mean Score
Within-project dataset	Deepcom	15.0%	29.0%	27.0%	16.0%	13.0%	100	2.84
	Deepcom+RS	14.8%	28.2%	28.0%	16.4%	12.6%	68	2.83
	Deepcom+QAcom	**8.8%**	**10.3%**	**22.1%**	**30.9%**	**27.9%**	68	**3.58**
	Rencos	24.0%	24.0%	25.0%	15.0%	12.0%	100	2.67
	Rencos+RS	23.6%	26.0%	25.4%	14.3%	10.7%	66	2.63
	Rencos+QAcom	**9.1%**	**13.6%**	**21.2%**	**25.8%**	**30.3%**	66	**3.55**
	NMT	30.0%	21%	23%	14.0%	11.0%	100	2.52
	NMT+RS	28.3%	22.6%	22.4%	14.5%	12.2%	60	2.59
	NMT+QAcom	**11.9%**	**15.8%**	**14.6%**	**30.1%**	**27.6%**	60	**3.46**
Cross-project dataset	Deepcom	24.0%	18.0%	29.0%	15.0%	14.0%	100	2.77
	Deepcom+RS	24.4%	20.2%	30.4%	15.3%	9.7%	42	2.66
	Deepcom+QAcom	**6.3%**	**11.0%**	**31.3%**	**28.1%**	**24.3%**	42	**3.56**
	Rencos	28.0%	22.0%	25.0%	15.0%	10.0%	100	2.57
	Rencos+RS	28.9%	21.3%	26.4%	15.6%	7.8%	35	2.52
	Rencos+QAcom	**8.9%**	**4%**	**30.1%**	**33.6%**	**23.4%**	35	**3.58**
	NMT	38.0%	26%	19%	9.0%	8.0%	100	2.23
	NMT+RS	37.8%	26.4%	18.6%	10.3%	6.9%	31	2.22
	NMT+QAcom	**17.9%**	**19.3%**	**35.6%**	**13.1%**	**14.1%**	31	**2.86**

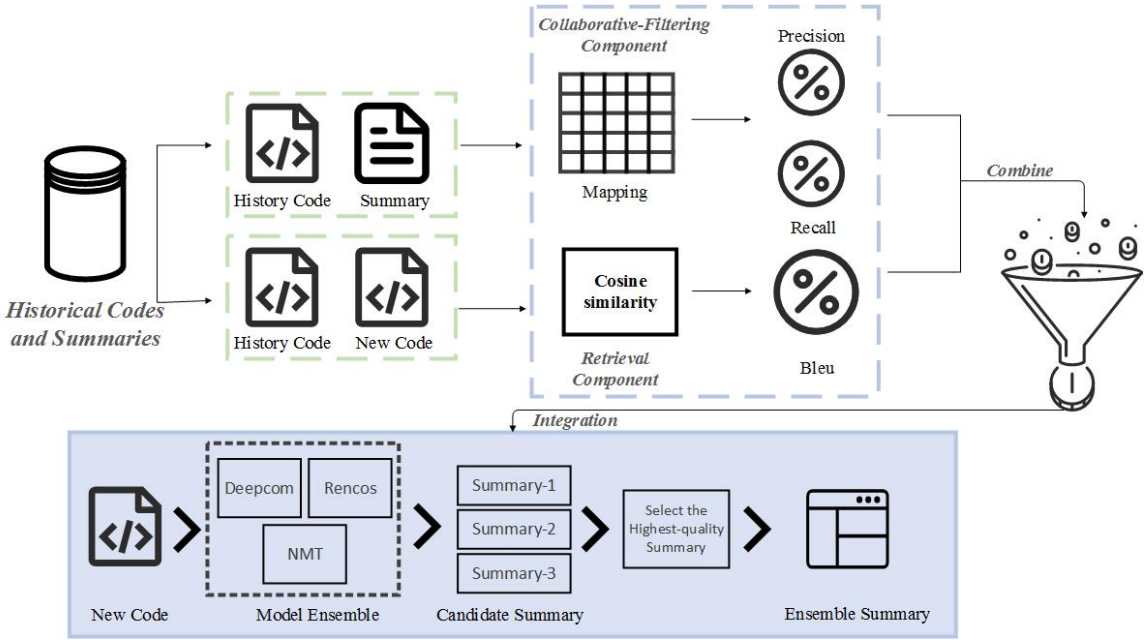

Fig. 4: The overview of Ensum.

TABLE V: The Evaluation Results of QAcom about Predicting Low-quality and High-quality summaries.

Dataset	Approach	Low-quality summaries		High-quality summaries	
		Precision	Recall	Precision	Recall
Within-project dataset	Deepcom+RS	71.6%	32.1%	29.0%	68.0%
	Deepcom+QAcom	**83.0%**	**44.3%**	**58.8%**	**72.5%**
	Rencos+RS	74.3%	33.8%	25.0%	66.0%
	Rencos+QAcom	**84.8%**	**46.2%**	**56.1%**	**72.9%**
	NMT+RS	74.1%	35.3%	26.7%	64.1%
	NMT+QAcom	**82.3%**	**42.1%**	**57.7%**	**72.2%**
Cross-project dataset	Deepcom+RS	71.0%	62.8%	25.1%	36.2%
	Deepcom+QAcom	**88.0%**	**78.1%**	**52.4%**	**75.9%**
	Rencos+RS	75.7%	67.3%	23.4%	32.7%
	Rencos+QAcom	**83.5%**	**77.1%**	**57%**	**73.0%**
	NMT+RS	83.7%	68.6%	17.2%	31.4%
	NMT+QAcom	**93.1%**	**79.4%**	**27.2%**	**49.6%**

TABLE VI: The Results of Complementarity Analysis.

Dataset	Approach	Good	Good (Only)	Good (All)
Within-project dataset	Deepcom	68	14	40
	Rencos	66	17	
	NMT	60	8	
Cross-project dataset	Deepcom	42	7	30
	Rencos	35	5	
	NMT	31	2	

approaches. Subsequently, Ensum will automatically select the best quality summary among the code summaries generated by the three summarization approaches for each code. Specifically, for each code, the three summarization approaches will generate three different summaries. We use CF scores and retrieval scores to compare these three different summaries.

We keep the highest-scoring summary as the corresponding summary for the current code. In this way, we can generate a summary of the best quality code for all codes.

C. Evaluation of Ensum

To automatically evaluate the effectiveness of Ensum, we also use metrics such as BLEU-4, METEOR, ROUGE-L. For example, given a dataset, we will use Ensum to integrate the three summarization approaches to obtain summaries of the corresponding test set in that dataset. Note that the Preserved-Ratio here is 100% because we generate summaries of the best quality for the entire test set by approach ensemble, rather than simply filtering. Finally, we compare the summaries generated by Ensum with the original three summaries generated by the three summarization approaches. If the final result is better than all three approaches, we successfully integrate the advantages of the three summarization approaches.

D. Experimental Results

1) Effectiveness of Ensum: As shown in Table VII, the ensemble results outperforms the results of the three summarization approaches, and the Preserved-Ratio reached 100%. For example, on the Within-project dataset, the BLEU-4, METEOR, ROUGE-L of ensemble results reached 0.406, 0.289, and 0.557, respectively. The results have reached the standard of high-quality summaries. Meanwhile, compared with the best of the three approaches, the improvement in BLEU-4, METEOR, ROUGE-L was 25%, 16%, and 9%, respectively, indicating the advantages of different aspects the three approaches are successfully integrated.

2) Results of Ensum Compared to Codesum: Codesum is the state-of-the-art ensemble approach for code summarization proposed by Chen et al. [17]. It classifies code comments into six categories by manual labeling, then trains the classifier to find a suitable code summarization approach for each type, and finally integrates them. Codesum needs to label and classify 20,000 comments to train the classifier manually, so it is a supervised method.

To compare the effectiveness of Ensum and Codesum in terms of approach ensemble, we first need to apply the three selected code summarization approaches: Deepcom, Rencos, and NMT on six categories, and thus find out the most suitable code summarization approach for each of the six categories. As shown in table VIII, for the categories "What", "How-it-is-done", and "How-to-use", Deepcom is the most suitable approach. For the categories "Why" and "Others", Rencos is the most suitable approach. For the "Property" category, NMT is the most suitable approach. Here, although for the category "Property", NMT does not have the highest BLEU-4 score, METEOR and ROUGE-L are the highest, so we chose the NMT approach for the "Property" category.

Subsequently, we evaluate the ensemble effectiveness of Ensum and Codesum on two datasets. As shown in Table VII, on the Within-project dataset, Ensum's effectiveness on BLEU-4, METEOR, ROUGE-L is improved by 26%, 17%, and 9% compared to Codesum, respectively. On the Cross-project dataset, Ensum improved 11%, 6%, and 5% on BLEU-4, METEOR, and ROUGE-L, respectively, compared to Codesum. For the current function, Ensum generates an entirely consistent summary with the ground truth compared to the results generated by Codesum. Thus, the idea of Ensum can gather the advantages of the approach more effectively and produce higher-quality code summaries than Codesum.

Ensum can generate high-quality code summaries by combining the advantages of three state-of-the-art code summarization approaches. Also, Ensum is more effective in taking the advantages of existing code summarization approaches for ensemble than Codesum.

VI. Threats to Validity

We discuss the threats to validity. One threat to validity concerns the actual results of the three code summarization approaches. To obtain the results, we experiment with their publicly available code and follow the descriptions in their papers or public code to set their hyper-parameters. In addition, some approaches require only the source code tokens, while others require the extraction of the source code AST, which makes it more challenging to reproduce the approaches. However, we keep in touch with the authors via email or GitHub issue to address the existing problems. The second threat to validity is about reproducing the results of Codesum. To mitigate this threat, we keep in touch with the authors and use the original code provided by the author to maintain correctness.

VII. Related Work

In this section, we introduce the related work about code summarization and quality assurance for document generation.

A. Code Summarization

Studies related to code summarization fall into three main categories: rule-based, retrieval-based, and neural model-based, and their interactions [4]. Rule-based approaches [24]–[27] generate code summaries through predefined rules. Sridhara et al. [25], [26] propose a Software Word Usage Model (SWUM) approach that selects statements from java methods. Moreno et al. [27] also predefine heuristic rules to generate summaries. Liu et al. [24] propose a latent semantic indexing and clustering method.

However, the rule-based approach has the limitation that it cannot handle cases where the rules are not predefined. Therefore, information retrieval-based approaches [28]–[33] are proposed where researchers consider more contextual information [27], [34]–[36]. Haiduc et al. [29] generate code summaries by retrieving similar term-based summaries. Wong et al. [32] propose CloCom for automatically generating code summaries by analyzing existing software repositories. Moskowitz-Attias et al. [30] use topic models and n-grams to predict the summaries corresponding to Java code. Brian et al. [28] propose a new topic modeling-based approach to source code summarization. Peige et al. [31] present an eye-tracking

TABLE VII: The Results of Ensemble Approach Ensum and Codesum.

Approach	Within-Project dataset				Cross-Project dataset			
	Bleu-4	Meteor	Rouge-L	Preserve Ratio	Bleu-4	Meteor	Rouge-L	Preserve Ratio
Deepcom	0.326	0.249	0.513	100%	0.205	0.158	0.344	100%
Rencos	0.281	0.210	0.458	100%	0.210	0.162	0.382	100%
NMT	0.255	0.197	0.457	100%	0.165	0.137	0.349	100%
Codesum	0.323	0.246	0.511	100%	0.203	0.163	0.382	100%
Ensum	**0.406**	**0.289**	**0.557**	100%	**0.226**	**0.172**	**0.403**	100%

TABLE VIII: Performances of the code summarization Models in Each Comments Category

Approach	Category	BLEU-4	METEOR	ROUGE
Deepcom	**What**	**0.225**	**0.189**	**0.375**
	Why	0.169	0.147	0.278
	How-is-it-done	**0.250**	**0.177**	**0.362**
	How-to-use	**0.234**	**0.178**	**0.354**
	Property	0.199	0.103	0.309
	Others	0.186	0.156	0.264
Rencos	What	0.135	0.168	0.324
	Why	**0.200**	**0.162**	**0.330**
	How-is-it-done	0.110	0.142	0.221
	How-to-use	0.196	0.141	0.234
	Property	0.145	0.094	0.298
	Others	**0.233**	**0.187**	**0.340**
NMT	What	0.102	0.101	0.309
	Why	0.029	0.060	0.164
	How-is-it-done	0.046	0.067	0.198
	How-to-use	0.068	0.077	0.21
	Property	**0.148**	**0.120**	**0.324**
	Others	0.133	0.185	0.288

study to produce a tool. Wong et al. [33] propose an approach that mines comments from the Question and Answer site.

In recent years, many neural network models [6], [17], [19], [37]–[47] have been proposed to generate code summaries. Iyer et al. [6] is the first to apply neural network techniques to code summary generation. Allaman et al. [19] use attention-based convolutional networks to generate short methods and class names as code summaries. Chen et al. [37] propose a framework BVAE to generate code summaries by learning vector representations of code and summaries. Shrivastava [45] proposes an automatic approach based on benchmarked and custom datasets. Arthur [46] proposes a novel system to automate the source code documentation process for C programming language . Mcburney and Mcmillan [47] present a technique that includes context by analyzing how the Java methods are invoked. Wan et al. [38] improve the encoder-decoder-based approach using a hybrid encoder and a reinforcement learning-based decoder to generate code summaries. Similarly, Wang et al. [39] use reinforcement learning a hierarchical attention network to combine multiple code features. In addition, Alon et al. [40], [41] propose a method for summary generation by representing source code by tokens and AST paths. Liang et al. [42] use the structural information of source code and RNN-based networks to generate code summaries. Ahmad [43] et al. use AST information and learn code representation by Transformer. LeClair et al. [44] propose a neural model called FunCom that combines tokens and AST of codes to generate summaries.

In addition, Chen et al. [17] propose a comment classifier. It classifies code summaries by manual labeling, then trains the classifier to find suitable code summarization approaches for each category. It is a supervised method. In contrast, Ensum does not require human-labeled scores and is therefore unsupervised and automated.

B. Quality Assurance for Document Generation

Our proposed approach is inspired by document generation quality assurance in the field of natural language processing [11], [12], [20], [48]–[50]. Zheng et al. [12] propose an approach to test the results of NMT translations. He et al. [50] propose a test approach for guaranteeing the quality of NMT with invariant structure. Tu et al. [48] propose coverage-based NMT to improve translation quality. Wu et al. [49] present Google's Neural Machine Translation system to reduce translation errors.

In addition, Jiang et al. [20] propose a quality assurance filter for commit message generation which is a learning-based approach. Subsequently, Wang et al. [11] propose an approach for automatic quality assurance of commit messages that uses collaborative filtering and retrieval methods. It worth mentioning that we are the first to apply quality assurance to code summarization to the best of our knowledge.

VIII. Conclusion

In this paper, we aim to improve code summarization through quality assurance approaches for document generation (QA4DG approaches). We first conduct an empirical study to investigate whether QA4DG approaches can be applied to code summarization. We find that QA4DG approaches are able to improve the practicability of existing code summarization approaches by predicting and filtering out low-quality code summaries generated by them. Then, inspired by the idea of QA4DG, we propose an ensemble code summarization approach named Ensum, which can integrate multiple single-model code summarization approaches by predicting the quality of their generated summaries. We evaluate the effectiveness of Ensum on two datasets, and the experimental results show that Ensum outperforms its used single-model code summarization approaches and a state-of-the-art ensemble code summarization approach.

IX. Acknowledgment

This work is supported by the National Key Research and Development Project (No. 2018YFB2101200), the National Natural Science Foundation of China (No. 62002034), the Fundamental Research Funds for the Central Universities (Nos. 2020CDJQY-A021, 2020CDCGRJ072) and the National Defense Basic Scientific Research Program (No. WDZC20205500308).

REFERENCES

[1] X. Xia, L. Bao, D. Lo, Z. Xing, A. E. Hassan, and S. Li, "Measuring program comprehension: a large-scale field study with professionals," in *Proceedings of the 40th International Conference on Software Engineering*, 2018, pp. 584–584.

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

[3] Q. Chen, H. Hu, and Z. Liu, "Code summarization with abstract syntax tree." in *International Conference on Neural Information Processing*, 2019, pp. 652–660.

[4] J. Zhang, X. Wang, H. Zhang, H. Sun, and X. Liu, "Retrieval-based neural source code summarization," in *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, 2020, pp. 1385–1397.

[5] X. Hu, G. Li, X. Xia, D. Lo, and Z. Jin, "Deep code comment generation," in *2018 IEEE/ACM 26th International Conference on Program Comprehension (ICPC)*, 2018, pp. 200–210.

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

[7] G. Klein, Y. Kim, Y. Deng, J. Senellart, and A. M. Rush, "Opennmt: Open-source toolkit for neural machine translation," in *Proceedings of ACL 2017, System Demonstrations*, 2017, pp. 67–72.

[8] "Interpretation of bleu score," https://cloud.google.com/translate/automl/docs/evaluate, 2020.

[9] Z. Liu, X. Xia, A. E. Hassan, D. Lo, Z. Xing, and X. Wang, "Neural-machine-translation-based commit message generation: how far are we?" in *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, 2018, pp. 373–384.

[10] "The shallowness of google translate," 2018.

[11] B. Wang, M. Yan, Z. Liu, L. Xu, X. Zhang, and D. Yang, "Quality assurance for automated commit message generation," *International Conference on Software Analysis, Evolution and Engineering*, 2021.

[12] W. Zheng, W. Wang, D. Liu, C. Zhang, Q. Zeng, Y. Deng, W. Yang, P. He, and T. Xie, "Testing untestable neural machine translation: an industrial case," in *2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*, 2019, pp. 314–315.

[13] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl, "Item-based collaborative filtering recommendation algorithms," in *Proceedings of the 10th international conference on World Wide Web*, 2001, pp. 285–295.

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

[15] S. Banerjee and A. Lavie, "Meteor: An automatic metric for mt evaluation with improved correlation with human judgments," in *Proceedings of the ACL Workshop on Intrinsic and Extrinsic Evaluation Measures for Machine Translation and/or Summarization*, 2005, pp. 65–72.

[16] C. Lin, J. Gao, G. Cao, and J. Nie, "Automatic evaluation of summaries," in *Workshop on Text Summarization Branches Out at ACL*.

[17] Q. Chen, X. Xia, H. Hu, D. Lo, and S. Li, "Why my code summarization model does not work," in *ACM Transactions on Software Engineering and Methodology*, 2021, pp. 1–29.

[18] "Our replication package," https://github.com/cqu-isse/Ensum, 2021.

[19] M. Allamanis, H. Peng, and C. A. Sutton, "A convolutional attention network for extreme summarization of source code," in *Proceedings of The 33rd International Conference on Machine Learning*, 2016, pp. 2091–2100.

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

[21] M.-T. Luong, H. Pham, and C. D. Manning, "Effective approaches to attention-based neural machine translation," in *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, 2015, pp. 1412–1421.

[22] "javalang," https://pypi.org/project/javalang.

[23] J. Vesterstrom and R. Thomsen, "A comparative study of differential evolution, particle swarm optimization, and evolutionary algorithms on numerical benchmark problems," in *Proceedings of the 2004 Congress on Evolutionary Computation (IEEE Cat. No.04TH8753)*, vol. 2, 2004, pp. 1980–1987.

[24] Y. Liu, X. Sun, X. Liu, and Y. Li, "Supporting program comprehension with program summarization," in *Computer and Information Science (ICIS), 2014 IEEE/ACIS 13th International Conference on*, 2014, pp. 363–368.

[25] G. Sridhara, E. Hill, D. Muppaneni, L. Pollock, and K. Vijay-Shanker, "Towards automatically generating summary comments for java methods," *Proceedings of the IEEE/ACM international conference on Automated Software Engineering*, pp. 43–52, 2010.

[26] G. Sridhara, L. Pollock, and K. Vijay-Shanker, "Generating parameter comments and integrating with method summaries," *2011 IEEE 19th International Conference on Program Comprehension*, pp. 71–80, 2011.

[27] L. Moreno, J. Aponte, G. Sridhara, A. Marcus, L. Pollock, and K. Vijay-Shanker, "Automatic generation of natural language summaries for java classes," in *2013 21st International Conference on Program Comprehension (ICPC)*, 2013, pp. 23–32.

[28] B. P. Eddy, J. A. Robinson, N. A. Kraft, and J. C. Carver, "Evaluating source code summarization techniques: Replication and expansion," in *2013 21st International Conference on Program Comprehension (ICPC)*, 2013, pp. 13–22.

[29] S. Haiduc, J. Aponte, L. Moreno, and A. Marcus, "On the use of automated text summarization techniques for summarizing source code," *2010 17th Working Conference on Reverse Engineering*, pp. 35–44, 2010.

[30] D. Movshovitz-Attias and W. W. Cohen, "Natural language models for predicting programming comments," in *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, 2013, pp. 35–40.

[31] P. Rodeghero, C. Liu, P. W. McBurney, and C. McMillan, "An eye-tracking study of java programmers and application to source code summarization," *IEEE Transactions on Software Engineering*, vol. 41, no. 11, pp. 1038–1054, 2015.

[32] E. Wong, T. Liu, and L. Tan, "Clocom: Mining existing source code for automatic comment generation," in *2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, 2015, pp. 380–389.

[33] E. Wong, J. Yang, and L. Tan, "Autocomment: mining question and answer sites for automatic comment generation," in *Proceedings of the 28th IEEE/ACM International Conference on Automated Software Engineering*, 2013, pp. 562–567.

[34] J. Fowkes, P. Chanthirasegaran, R. Ranca, M. Allamanis, M. Lapata, and C. Sutton, "Tassal:autofolding for source code summarization," in *International Conference on Software Engineering Companion*, pp. 649–652.

[35] P. W. McBurney, "Automatic documentation generation via source code summarization," in *Proceedings of the 37th International Conference on Software Engineering - Volume 2*, vol. 2, 2015, pp. 903–906.

[36] P. W. McBurney and C. McMillan, "Automatic source code summarization of context for java methods," *IEEE Transactions on Software Engineering*, vol. 42, no. 2, pp. 103–119, 2016.

[37] Z. M. Chen Q, "A neural framework for retrieval and summarization of source code," in *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, 2018, pp. 826–831.

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

[39] W. Wang, Y. Zhang, Y. Sui, Y. Wan, Z. Zhao, J. Wu, P. Yu, and G. Xu, "Reinforcement-learning-guided source code summarization via hierarchical attention," *IEEE Transactions on Software Engineering*, no. 1, pp. 1–1, 2020.

[40] U. Alon, S. Brody, O. Levy, and E. Yahav, "code2seq: Generating sequences from structured representations of code," in *International Conference on Learning Representations*, 2018.

[41] U. Alon, M. Zilberstein, O. Levy, and E. Yahav, "code2vec: learning distributed representations of code," *Proceedings of the ACM on Programming Languages*, vol. 3, p. 40, 2019.

[42] Y. Liang and K. Q. Zhu, "Automatic generation of text descriptive comments for code blocks." in *Association for the Advance of Artificial Intelligence*, 2018, pp. 5229–5236.

[43] W. U. Ahmad, S. Chakraborty, B. Ray, and K.-W. Chang, "A transformer-based approach for source code summarization," in *Proceed-*

ings of the 58th Annual Meeting of the Association for Computational Linguistics, 2020, pp. 4998–5007.

[44] A. LeClair and C. McMillan, "Recommendations for datasets for source code summarization," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, 2019, pp. 3931–3937.

[45] P. Shrivastava, "Neural code summarization." *arXiv preprint arXiv:2103.01025*, 2021.

[46] M. P. Arthur, "Automatic source code documentation using code summarization technique of nlp," *Procedia Computer Science*, vol. 171, pp. 2522–2531, 2020.

[47] P. W. McBurney and C. McMillan, "Automatic documentation generation via source code summarization of method context," in *Proceedings of the 22nd International Conference on Program Comprehension*, 2014, pp. 279–290.

[48] Z. Tu, Z. Lu, Y. Liu, X. Liu, and H. Li, "Modeling coverage for neural machine translation," in *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, vol. 1, 2016, pp. 76–85.

[49] Y. Wu, M. Schuster, Z. Chen, Q. V. Le, and M. Norouzi, "Google's neural machine translation system: Bridging the gap between human and machine translation," *arXiv preprint arXiv:1609.08144*, 2016.

[50] P. He, C. Meister, and Z. Su, "Structure-invariant testing for machine translation," in *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, 2020, pp. 961–973.