Re_Trans: Combined Retrieval and transformer model for Source Code Summarization

Chunyan Zhang
State Key Laboratory of Mathematical Engineering and Advanced Computing

Qinglei Zhou
Zhengzhou University

Meng Qiao
State Key Laboratory of Mathematical Engineering and Advanced Computing

Ke Tang
State Key Laboratory of Mathematical Engineering and Advanced Computing

Lianqiu Xu
State Key Laboratory of Mathematical Engineering and Advanced Computing

Fudong Liu (lwfydy@126.com)
State Key Laboratory of Mathematical Engineering and Advanced Computing

Research Article

Keywords: Source code summarization, Program analysis, Information retrieval, Deep learning

Posted Date: June 22nd, 2022

DOI: https://doi.org/10.21203/rs.3.rs-1742575/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License
Re_Trans: Combined Retrieval and transformer model for Source Code Summarization

Chunyan Zhang¹, Qinglei Zhou², Meng Qiao¹†, Ke Tang¹, Lianqiu Xu¹ and Fudong Liu¹*

¹*School of Information Engineering, State Key Laboratory of Mathematical Engineering and Advanced Computing, 62 Science Avenue, ZhengZhou, 450000, Henan, China.
²School of Information Engineering, ZhengZhou University, 100 Science Avenue, ZhengZhou, 450000, Henan, China.

*Corresponding author(s). E-mail(s): lwfydy@126.com; Contributing authors: ieczhang@163.com; ieqlzhou@zzu.edu.cn; kopo.code@gmail.com; tuck3r@foxmail.com; lianqiuxu@163.com;
†These authors contributed equally to this work.

Abstract
Source code summarization (SCS) refers to the natural language description on what a source code is performing. It can help developers understand programs and maintain software efficiently. Retrieval methods (RM) generate SCS reorganize terms selected from source code or use SCS of similar code snippets. Generative methods (GM) generate SCS via attentional encoder-decoder architecture. However, a GM can generate SCS for any code, but sometime the accuracy is still far from expectation (due to the lack of numerous high-quality training set). A RM is considered with a higher accuracy, but usually fail to generate SCS for a source code in the absence of a similar candidate in the database. In order to effectively combine the advantages of RM and GM, we propose a new method Re_Trans. For a given code, we first utilize RM to get the most similar code w.r.t semantic and its SCS (S_RM). Then, we input the given code and similar code into the trained discriminator. If the discriminator outputs 1, we take S_RM as the result, otherwise, we utilize the GM, transformer, to generate the given code' SCS. Particularly, we use AST-augmented and code sequence-augmented information to make the
source code semantic extraction more fully. Furthermore, we build a new SCS retrieval library through the public dataset. We evaluate our method on a data set of 2.1 million Java code-comment pairs, and experimental results show improvement over the state-of-the-art (SOTA) benchmarks, which demonstrates the effectiveness and efficiency of our method.

**Keywords:** Source code summarization, Program analysis, Information retrieval, Deep learning.

## 1 Introduction

Source code summarization (SCS), also named code comment, is a term coined by Haiduc et al. [1]. It is a natural language description of programming fragment. Program maintenance is the most expensive and time-consuming stage in the software life cycle [2]. High-quality SCS is essential to program comprehension and maintenance, which can help developers save time spent on navigating source code and understand programs quickly. Unfortunately, with the rapid update of software, most SCS is mismatched, outdated, and missing. Thence, SCS generation has been researched extensively and made lots of remarkable achievements [3–13].

SCS generation is a hot field that emerged more than a decade ago. Its methods can be divided into three categories: manually-crafted template, information retrieval-based (IR-based), and Deep Learning-based (DL-based). The manual template methods usually extract keywords from source code to generate SCS [14–16]. But it misses a lot of potential information of the source code. The IR-based methods are widely used in SCS generation. It generates SCS usually by searching keywords from the given code or code comments of the code that is most similar to the given code. For example, Haiduc et al. [3, 17] analyzed source code using the VSM and LSI methods, producing natural language description of the classes or methods. Li et al. [18] used the Latent Dirichlet Allocation (LDA) technology to conduct topic mining on resources, such as code, documentation, question and answer information, and automatically generated code topic summarization. Wong et al. [19] utilized code clone detection technology to find the code snippet with similar syntax from the existing code bases, and applied summarization to other codes with similar syntax. However, IR-based methods over-reliance on identifier naming and the similar amount of source code in the dataset.

Currently, almost all works use the DL-based method in SCS generation task. The common DL techniques include recurrent neural network (RNN) [20] and its variant models, convolution neural network (CNN) [21] and its variant models, transformer model [22], and large-scale language training model (e.g., BERT [23] and GPT [24]). The Attention mechanism is usually used as a key auxiliary to the above methods. Deep-Com [8] utilized a seq2seq model to generate SCS of Java method based on Attention mechanism. Notably, its
SBT method (a tree traversal way) made a major breakthrough in structure information extraction and adopted by many works. For example, in 2020, the Hybrid-DeepCom [25] extended the work of [8]. It combined the source code sequence and the SBT sequences to generate SCS. Especially, the camel case naming was used to solve out-of-vocabulary identifiers problem. LeClair et al. [26] improved the accuracy of SCS by processing source code AST information on the basis of ast-attendgru [10]. Wang et al. [12] and Uddin Ahmad et al. [13] utilized transformer to generate the SCS, and improved the effectiveness and accuracy comparing with existing methods. The effect of work [26] and [13] is higher than other works above. The main reason is that the paper [26] takes the whole AST as a graph to represent structure information instead of AST-sequences or AST-paths, which preserves the structure information more completely. Hence, we use this AST embedding way in our method. Besides, the paper [13] proves that the transformer model performs well in SCS generation task. However, these methods use either IR-based or Neural Machine Translation (NMT)-based methods to generate SCS. NMT-based methods are generative methods. A GM can generate SCS for any code, but the result is still far from expectation due to the absence of high-quality training set. A RM has high accuracy, but it requires a similar candidate in the database to the given code.

In this paper, for the purpose of combining the advantages of RM and GM, we propose a neural approach to generate SCS, Re_Trans. Our method contains two SCS generation models, retrieval-based and transformer. For a given code, we first utilize RM to get the most similar code w.r.t semantic and its SCS (S_RM). Then, we input the given code and similar code into the trained discriminator. If the output is 1, we take S_RM as the result, otherwise, we utilize the transformer model to generate the final SCS. Re_Trans adopts a suitable SCS generation model to the given code.

Particularly, the augmented information makes the source code semantic extraction more fully: 1) We use AST to represent the structure information and enhance it by adding data flow and control flow edges to AST. Moreover, we utilize GCN to encode the whole AST for preserving the structure information more completely. 2) We use code sequence to represent the syntax information and enhance it by adding position information to code. In our retrieval model, we adopt BiGRU to encode sequences, and choose self-attention mechanism to encode it in our transformer model. We conduct experiments on a popular real-word dataset, and the results demonstrate that our method outperforms the SOTA work with widely-used metrics (BLEU, METEOR and ROUGE). Furthermore, we also perform the time-consuming experiments to confirm the efficiency of our method.

The main contributions of this paper are as follows:

- We propose a Re_Trans system by combining retrieval and generative methods, and adopt the suitable SCS generation model for a given source code.
We use non-leaf nodes of the AST to build a directed graph, and enhances the edge information though data flow and control flow. To the best of our knowledge, this is the first time that such an efficient structure representation mode has been used in SCS task.

We perform extensive experiments on a public real-world dataset. All results confirm that the Re_Tran is effective and outperforms the SOTA methods.

2 Background

This section introduces the technical knowledge used in our method: Bidirectional GRU (BiGRU) [27], GCN [28], transformer [22] and beam search [29]. Re_Tran contains one retrieval-based model and one generative model, and uses a discriminator to decide which model’s result is the final SCS for the give code. In retrieval model, we use GCN to encode ASTs and BiGRU to encode code sequences, and get SCS by calculating the similarity between code semantics. In generative model, we utilize transformer to get SCS. Moreover, we use a beam search algorithm in Re_Tran to ensure that the generated SCS is non-random and closest to the real result.

3 Our Approach

3.1 Overview

The workflow of our proposed method (Re_Tran) is shown in Figure 1. For a given code, we first use the retrieval-based model to get its most similar code w.r.t semantic. Then, we input the given code and its similar code into the discriminator. If the output is 1, we take the similar code’s SCS as the result. Otherwise, we utilize transformer model to get the result.

Re_Tran mainly contains three steps: 1) Data representation (see Section 3.2). Re_Tran parses the source code into AST and source code sequence, and processes code summarization by a simple text processing method. 2)
Fig. 2 Overall framework of Re_Trans.

Model training (see Section 3.3). Re_Trans includes a retrieval-based model (see green box), a generative model (see blue box) and a discriminator (see section 3.5). 3) Model testing (see Section 3.4).

3.2 Data processing

In this paper, we use a large public dataset <Java code, comment> pairs. Our data processing way is available in various programing languages. We represent the Java code as parsed AST and code sequence, and process comment into simple text.

For one sample, we show the source code structure information in Figure 2. Initially, we use the javalang toolkit to parse source code into an AST and only keep its non-leaf nodes. There are two reasons for removing the leaf nodes: 1) Avoid repeated processing. Because the leaf nodes correspond to the source code text, which has been processed in code sequence information. 2) Non-leaf nodes represent the source code structure information to a certain extent, and this structure saves much traversal time.

Furthermore, we enhance source code semantic information by adding data flow and control flow to AST referring to the work of Wang et al. [29]. Considering our AST without leaf nodes, in data flow information, we connect a node to its next brother node (from left to right). It solves the problem that graph neural networks do not consider the order of nodes. For example, the green lines in the red box (aa) in Figure 4 connect three sibling nodes of "Modifier", "FormaParameter" and "ForStatement". The added edges are (Modifier, FormaParameter) and (FormaParameter, ForStatement). In control flow information, we select "IfStatement", "WhileStatement", "ForStatement" and "BlockStatement" nodes. "BlockStatement" is the root node of the code block when executing source code sequentially. According to the characteristics of each node, we connect their child nodes to form new edges.
Finally, we utilize depth-first traversal to obtain the edge set of the directed AST. The edge set is \( e = (e_1, e_2, \ldots, e_n) \), where \( n \) is the number of edges. For any edge \( e_{ij} = (t_i, t_j) \), \( t_i, t_j \) indicate the start node and end node. Compared with undirected AST, directed AST can represents the sequence structure information more accurately. We take the edge information as an initialization vector and input it into GCN for semantic extraction of source code.

The source code sequence information is shown in Figure 3. Firstly, we treat each source code as a plain text (as shown in blue box), each word corresponds to a unique identifier by dictionary mapping. Secondly, since the self-attention mechanism ignores the position information of the input data, we add row and column position information to the source code. For the row position information, we assign integer values starting from zero to each row of Java function sequentially (as shown in red box). For the column position information, we assign integer values starting from zero in the word order of each code (as shown in green box). Finally, we concatenate the three initialization vectors to form a new vector. For example, for the word “static”: the unique identifier initial vector is \( \text{Emb}_1 \), the row position initial vector is \( \text{Emb}_2 \), and the column position initial vector is \( \text{Emb}_3 \). The initial vector of “static” is \( \text{Emb} = \text{cat}(\text{Emb}_1, \text{Emb}_2, \text{Emb}_3) \).

Code comment is similar to text in NLP. We perform simple serialization, which assigns unique identifiers to all words in the comment. After initialization, we take the comment as part of the input to the transformer model.

### 3.3 Model Training

Re_Trans contains two SCS generation models: retrieval model and transformer. We show them in Figure 4 and Figure 5.

The goal of the retrieval model is to match the code snippet that is most similar to the given code and its SCS. Different codes may implement the same function, which is difficult to judge whether the codes are similar by simply comparing the code text. The semantic similarity perfectly solves this issue. Thereby, we compare the code’ semantic similarity in retrieval-based model. For a sample, we use GCN to process the enhanced AST, and BiGRU to deal with the enhanced code sequence. The semantic vector of sample is the results.
concatenation of GCN and BiGRU. We use Euclidean distance to find the code in retrieval library that is most similar to the sample and its SCS. The construction process of the retrieval library will be detailed in Section 4.1.

The goal of the transformer model is to generate new SCS for each input function. For a sample, we also use GCN to extract its structure information. As for the syntactic information, we directly utilize the transformer’s encoder. The sample semantic vector is the results concatenation of GCN and self-attention mechanism. We use the transformer’s decoder to convert the sample semantic vector to its SCS.

### 3.4 Model Test

In this section, we introduce the testing process of Re_Trans and show it in Figure 6. For a sample, we first use the RM to get the most similar code w.r.t semantic (S_code) and its SCS (S_RM). Then, we input the sample and S_code into discriminator to get the label \((\text{sim}\_\text{label})\). If \(\text{sim}\_\text{label} = 1\), we will take S_RM as the final result. Otherwise, we utilize the transformer model to get the result (S_GM). Particularly, the discriminator’ training process is detailed in Section 3.5.

### 3.5 Discriminator

In order to train the discriminator, we randomly sample 200,000 samples from the dataset in Section 4.1, and divide them into training and testing sets in a ratio of 8:2. The training process is shown in Figure 7.

First, we assign the label to all samples. For a sample: 1) We use the retrieval model in Section 3.3 to obtain its S_code and S_RM, and use transformer model to generate the its SCS (G_RM). 2) We utilize the cosine
distance to calculate the similarity between S_RM, G_RM and the target SCS, respectively. If S_RM is better, we set the sample is \langle\text{sample, S\_code, 1}\rangle. Otherwise, the sample is \langle\text{sample, S\_code, 0}\rangle. Repeat these two steps, we set all discriminator data in the form of \langle\text{sample, S\_code, label}\rangle, where label represents 0 or 1.

Second, we train the discriminator with 160,000 samples, and the parameters are shown in Section 4.3. Especially, we utilize MLP to calculate the semantic similarity between SCSs instead of pure string, which avoids the drawback of similar text but different meaning. Finally, we use the remaining 40,000 samples to test the trained discriminator.

4 Experiments setup

4.1 Dataset Analysis

The dataset contains around 2.1 million \langle\text{Java function, comment}\rangle pairs [30], which is widely used in lots of SCS generation tasks [10, 22, 31]. We analysis the dataset from two aspects: 1) Statistical length distribution of source codes and their comments (see Figure 8). 2) Count the scale of functions’ number with same comment (see Figure 9).

From Table 1 and Figure 8, we can see that the comment length distribution is relatively uniform, ranging from 3 to 13. Short comment helps people understand code’ function quickly. The code lengths are distributed between
1 and 100, which is approximately normal distribution. When code length is larger than 70, the number is almost unchanged, so we set 70 as the optimal input length parameter.

In order to build a retrieval library, we remove invalid data whose comment corresponds to only one function. In Figure 9(a), the function of scale 2 is close to 160,000, far exceeding the number of other scales. We denoise these functions and use them as a retrieval library. From Figure 9(b), we find that the number of scales over 80 is almost 1. Hence, we count the functions with scales from 10 to 80 to show the distribution of similar functions in the entire dataset more intuitively, as shown in Figure 9(c).

4.2 Hardware

We use a workstation with two Intel(R) Xeon(R) Gold 6154 CPU @ 3.00GHz, 128gb RAM, and two Titan XP GPUs. It is necessary to train on GPUs with 64gb VRAM due to the large size of our model and dataset.

4.3 Parameter settings

In this paper, we show the main parameter settings in Table 2. In transformer model, we use the Adam optimizer, and set epsilon to 1e-9 and \((\beta_1, \beta_2)\) to (0.9, 0.98). The transformer \((N=4, h=4, \text{dim}=256)\), where \(N\) is the number of encoder layer, \(h\) is the number of multi-head-attention, and \(\text{dim}\) is the embedding dimension. Particularly, we use the NoamOpt to obtain the learning
rate dynamically, where the warmup is 200, and factor is 1. The batch_size is set to 256, and epoch is 40. In retrieval model, we also use the Adam optimizer, and set the GCN and BiGRU layer to 2. Besides, we set the leaning rate to 1e-4, and the epoch to 30.

### 4.4 Metrics

Similar to some existing works [11, 13], we evaluate the performance of Re_Trans and baselines using widely-used metrics: BLEU [32], METEOR [33] and ROUGE [34]. Their scores are in the range [0, 1] and reported in percentages in this paper. For one Java function \( x \), suppose that the generated SCS by Re_Trans is \( y \), namely candidate sentences. The ground-truth SCS of \( x \) is \( s \), namely reference sentences.

BLEU measures the \( n \)-gram precision of candidate sentences that appear in reference sentences. The higher score of BLEU-N (\( N = 1, 2, 3, 4 \)), the higher quality of \( y \). The formula to calculate BLEU-N is as follows:

\[
BLEU - N_{(y,s)} = BP \cdot \exp \left( \sum_{n=1}^{N} w_n \cdot \log p_n \right),
\]

where \( BP \) is the brevity penalty, used to punish the short candidate sentences. \( p_n \) is the precision score of the \( n \)-gram matches between \( y \) and \( s \). \( w_n \) is usually the uniform weight of \( n \)-gram, \( w_n = 1/N \). \( l_y \) is the length of \( y \), and \( l_r \) is the length of \( s \).

ROUGE-L utilizes the Longest Common Subsequence (LCS) between \( y \) and \( s \). The formula to calculate ROUGE-L is as follows:

\[
ROUGE-L = \frac{1}{l_y} \cdot \left( \frac{l_y}{l_r} \right) ^{\frac{1}{n}}.
\]
\[ R_{\text{lcs}} = \frac{LCS(y, s)}{\text{len}(y)}, \quad P_{\text{lcs}} = \frac{LCS(y, s)}{\text{len}(s)}, \quad \text{ROUGE} - L = \frac{(1 + \beta^2) \cdot R_{\text{lcs}} \cdot P_{\text{lcs}}}{R_{\text{lcs}} + \beta^2 \cdot P_{\text{lcs}}} \]

where \( R_{\text{lcs}} \) is the recall rate and \( P_{\text{lcs}} \) is the precision rate, \( \beta = P_{\text{lcs}} / R_{\text{lcs}} \).

METEOR considers the precision and recall rate based on the entire corpus. It uses WordNet to expand the synonym set, which has a high correlation with human judgment. However, this metric only used in Java programming language. The formula to calculate METEOR is as follows:

\[
P = \frac{\text{mapped}}{\text{total}_y}, \quad R = \frac{\text{mapped}}{\text{total}_s}, \quad F_{\text{mean}} = \frac{10P \cdot R}{R + 9P}, \quad \text{Penalty} = 0.5 \cdot \left( \frac{\text{blocks}}{\text{unigrams}} \right)^3
\]

\[ \text{METEOR} = F_{\text{mean}} \cdot (1 - \text{Penalty}) \]

where \( \text{total}_s, \text{total}_y \) are the total number of words in \( s \) and \( y \), respectively. \( \text{mapped} \) represents the mapping result of words in \( y \) on \( s \), and only retains words that appear at most once in \( s \). \( \text{unigrams} \) represents a single word, \( \text{blocks} \) represents how well a phrase is matched. \( \text{Penalty} \) is used to avoid the case where only word matching is considered.

5 Results and Analysis

Our research objective is to determine the Re_Trans outperforms current baselines. We also want to demonstrate the efficiency of RGSGS and effectiveness of the retrieval library built by us. Notably, all methods are trained on the dataset described in Section 4.1. We answer the following Research Questions (RQs) to explore these situations:

RQ1: What is the performance of Re_Trans compared to the baselines?
RQ2: Why the Re_Trans approach performs well? RQ3: How is the high efficiency of Re_Trans? RQ4: How well the SCS retrieval library we built?

5.1 Baselines

To demonstrate the effectiveness of our method, we compare it with the SOTA methods in recent years. The baselines are described as follows:

(2019) ast-attendgru is an attentional encoder-decoder architecture to generate SCS [10]. It enhances the SBT and AST flattening procedure proposed by Hu et al. [7, 21] and shows a higher performance, so we only compare against this approach.

(2018) graph2seq [35] is a general neural encoder-decoder architecture that solves the graph-to-sequence problem. It achieves SOTA results on a SQL-natural language task using BLEU-4 metric. The opensource code of graph2seq is convenient to do comparative experiments.

(2020) ConvGNN [26] is a graph-based neural model architecture to SCS. Different from the flattened AST, it takes the AST as a graph and performs
well, which is closer to our method in terms of code structure information extraction.

(2020) transformer-based [13] is the first method of applying transformer model to SCS. It incorporates relative positional encoding and copies attention mechanism into transformer model to improve the SCS quality. We also use transformer model as the Re_Trans’ generative model.

(2020) Rencos is a novel retrieval-based neural approach that augmented an attentional encoder-decoder architecture in SCS [11]. It retrieves two most similar code snippets to the given code from aspects of semantic and syntax, respectively. Rencos enhances the accuracy of the SCS by fusing the retrieved results into the generative model.

(2020) Re2Com [36] is similar to Rencos, which also uses retrieval-based method to enhance the SCS’ accuracy. For a given code, Re2Com retrieves its most similar code and the SCS pair. Then it takes the given code (its code text and AST sequence), the most similar code and SCS as input to the encoder. Experiments demonstrate the effectiveness of this method.

5.2 Re_Trans vs Baselines

Among baselines, ast-attendgru, graph2seq, ConvGNN, and transformer-based belong to GM. Rencos and Re2Com are methods that combine retrieval and generative techniques. According to the parameter settings in Table 2, we show the experimental results of Re_Trans and baselines in Table 3.

From Table 3, we find that the effect of Rencos method is relatively poor. The reason may be that the retrieved two most similar syntax-layer and semantic-layer code snippets, which are still different from real code’ semantics. When taking them as input of the encoder, Rencos may produce biased SCS. Although ast-attendgru, Rencos and Re2Com use flattened AST to represent code structural information, the AST sequence is a linear problem in nature. The effective of ConvGNN, transformer-based and Re2Com are close to Re_Trans. The Re_Trans and ConvGNN use the similar source code semantic extraction method, AST graph and source code sequence, but the Re_Trans performs better. One reason is that our method enhances the AST and code sequence, which can extract the source code semantic information more fully (explained in Section 5.3). Besides, the paper [26] uses a decoder with attention mechanism, which is less effective than transformer model [30]. We also

<table>
<thead>
<tr>
<th>Methods</th>
<th>BLEU-1</th>
<th>BLEU-2</th>
<th>BLEU-3</th>
<th>BLEU-4</th>
<th>ROUGE-L</th>
<th>METEOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>ConvGNN[26]</td>
<td>39.21</td>
<td>22.50</td>
<td>15.73</td>
<td>11.97</td>
<td>40.25</td>
<td>20.12</td>
</tr>
<tr>
<td>Re2Com[36]</td>
<td>38.96</td>
<td>23.08</td>
<td>17.49</td>
<td>15.67</td>
<td>40.01</td>
<td>20.04</td>
</tr>
<tr>
<td><strong>Re_Trans</strong> (ours)</td>
<td>42.97</td>
<td>25.85</td>
<td>18.58</td>
<td>16.83</td>
<td>42.64</td>
<td>22.15</td>
</tr>
</tbody>
</table>
Table 4 The ablation experiment results.

<table>
<thead>
<tr>
<th>Methods</th>
<th>BLEU-1</th>
<th>BLEU-2</th>
<th>BLEU-3</th>
<th>BLEU-4</th>
<th>ROUGE-L</th>
<th>METEOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Seq</td>
<td>35.58</td>
<td>22.36</td>
<td>15.22</td>
<td>10.86</td>
<td>37.63</td>
<td>18.97</td>
</tr>
<tr>
<td>ast_aug+Seq</td>
<td>38.99</td>
<td>23.74</td>
<td>16.76</td>
<td>15.32</td>
<td>40.01</td>
<td>20.26</td>
</tr>
<tr>
<td><strong>ast_aug+Seq_aug</strong></td>
<td><strong>42.97</strong></td>
<td><strong>25.85</strong></td>
<td><strong>18.58</strong></td>
<td><strong>16.83</strong></td>
<td><strong>42.64</strong></td>
<td><strong>22.15</strong></td>
</tr>
<tr>
<td>ast_leaf_aug+Seq_aug</td>
<td>43.01</td>
<td>26.30</td>
<td>19.22</td>
<td>16.30</td>
<td>42.68</td>
<td>22.01</td>
</tr>
</tbody>
</table>

find that the BLEU-N of transformer-based method is on average about 7% higher than ConvGNN. The poor performance of graph2seq is that it only considers the structural information of source code, but ignores the syntactic information in SCS task.

In Table 3, the Re_Trans performs best, the main reasons are following: 1) we combine the code sequence-augmented and AST-augmented to characterize source code. 2) The generative model is transformer model. Transformer is generally better than the seq2seq architecture in all tasks. In addition, the BLEU-N gradually decreases as the N increases. It shows that there is still a lot of room for improvement in the long sequence matching between the generated SCS and target. Furthermore, it also indicates that the current SCS generation model still needs to be further studied and improved.

5.3 Ablation study

In this section, we will illustrate why the Re_Trans method works well through ablation experiments. The source code semantic representation ways mainly include source code sequence information (Seq), AST-augmented (ast_aug) combined with Seq, and AST-augmented combined with code sequence-augmented information (Seq_aug). However, we also test the SCS effect of preserving leaf nodes of AST-augmented (ast_leaf_aug) combined with Seq_aug. For these different semantic extraction methods of source code, we use Re_Trans to generate SCS, and show the ablation experiments results in Table 4.

In Table 4, we find that the SCS quality has been improved after the Seq combining with the ast_aug information. It is because pure sequence information ignores the potential and complex structural information of source code, and ast_aug preserves the structural information of source code. As we all know, the self-attention mechanism ignores position information when encoding sequence information. Thereby, we add position information to source code sequences to solve this problem. As shown in Table 4, the effect is significantly improved when we use “ast_aug+Seq_aug” to represent source code. The BLEU-1 is improved by 10.2%, and BLEU-4 is improved by 9.9%. Moreover, we also find that the effect of “ast_leaf_aug+Seq_aug” is better than “ast_aug+Seq_aug”, but the gap is slight. In Section 5.4, we demonstrate that the time efficiency of the latter is much higher than that of the former. Therefore, we choose the AST without leaf nodes to characterize the source code structure information.
5.4 High Efficiency

High efficiency is an important advantage of Re_Trans compared to other SCS methods. It is mainly reflected in two aspects: the efficiency of source code semantic extraction and the efficiency of Re_Trans’ generation model.

(1) Efficiency of source code semantic extraction: The efficiency of source code semantic extraction is mainly reflected in AST structure. Because the AST contains a large number of leaf nodes with irregular user-defined identifiers, which makes the data processing time-consuming. From Table 4, we know that the SCS’ result of AST with leaf nodes (ast_with_leaf) is close to that without leaf nodes (ast_no_leaf) in our method. In order to demonstrate the efficiency of source code semantic extraction, we randomly select 100,000 samples from the dataset, and construct AST structure information with and without leaf nodes respectively. We calculate the accumulated time (the time unit is seconds) for each 10,000 samples, and show the results in Figure 10(a). Furthermore, we randomly select 1,000 samples from these 100,000 samples, and test their SCS time through Re_Trans for these two source code structure representations respectively. We calculate the accumulated time (the time unit is seconds) for each 100 samples processed, and show the results in Figure 10(b).

![Graph](image1)

(a) Data processing time of AST with/no leaf nodes.

![Graph](image2)

(b) SCS time of Re_Trans’ generative model.

**Fig. 10** The efficiency of source code semantic extraction.

In Figure 10(a), the ast_no_leaf extraction method we proposed (the green line) takes significantly less time, and the growth rate is slow, which reflects the efficiency of AST without leaf nodes in data processing. In Figure 10(b), our proposed AST structure (the purple line) is more efficient than ast_with_leaf extraction method in SCS task. Therefore, we use the AST that removes leaf nodes, which not only saves the time of graph traversal, but also avoids the repeated operation of source code sequence information processing.

(2) Efficiency of the Re_Trans generation model:

In order to demonstrate the efficiency of the Re_Trans generation model, we randomly selected 1,000 samples from the dataset. We test the SCS generation time of Re_Trans’ generation model (Re_Trans_generative) on these data, Re_Trans, Re_Trans generative model with leaf nodes...
(Re_Trans_leaf_generative), Rencos and Rencos generative model (Rencos_only_NMT). The time statistic results are shown in Figure 11.

From Figure 11, we can see that Re_Trans_generative (the purple line) takes the shortest time and is more efficient than Rencos_only_NMT (the cyan line). Besides, the Rencos method (the blue line) takes the longest time, and the efficiency of our method (the green line) is higher than Rencos. The main reason is that Rencos requires both IR-based and NMT-based methods for each test. But Re_Trans only uses IR-based method, or IR-based and generative methods for each test. The above-mentioned generation model of Re_Trans is more efficient than Rencos. Furthermore, in our test, the Rencos retrieval database has 6648 samples and Re_Trans has 10000 samples, but their average retrieval time of one test data is about 0.104s. Obviously, the Re_Trans’ retrieval efficiency is higher. In practice, with the continuous expansion of retrieval database, the average retrieval time will increase accordingly. Because the test data needs to match each item in the retrieval database to find the sample with the highest similarity score.

5.5 SCS library

From the dataset analysis in Section 4.1, we find that the public dataset has a large number of similar functions, which are suitable for building a SCS retrieval library. In order to demonstrate the effectiveness of the retrieval library, we conduct the following experiments:

(1) In order to test the effectiveness of our retrieval library, we randomly select samples from the code retrieval library after data processing, and use the t-SNE data dimensionality reduction and visualization technology. The final result is shown in Figure 12.

In view of the fact that too many samples lead to a large number of repeated points in the picture, which affects the visualization effect. We select two groups of 100 and 200 random samples, corresponding to (a) and (b) in Figure 12. From Figure 11, we can see that almost all the two points with same color overlap or are very close. It indicates that functions with the same summarization still retain the same semantic information after data processing. Furthermore, it also shows the effectiveness of retrieval library built by us.
(2) In order to test the effectiveness of Re_Trans retrieval model, we randomly select 1,000 samples, and calculate the probabilities of Top $k$ ($p@k$) between test function and each sample by Euclidean distance, where $k$ is 1, 3, 5 and 10. The Top $k$ is to find the top $k$ numbers from the retrieval library. The higher $p@k$, the better matching effect. The test is carried out in 10 groups, and we use the boxplot to visual display. We show the result in Figure 13.

From Figure 13, we can see that the $p@k$ increases gradually with the increase of $k$. In 10 rounds of testing experiments, when $k = 3$, the minimum value is 80%, the maximum value is 96%, and the average value is 92%. For any test code snippet, it indicates that the same semantics’ code snippets searched from the retrieval library are the similar code snippets with higher accuracy. It also indicates the feasibility of using the SCS of similar code as the SCS of test code.

6 Conclusion and Discussion

In this paper, we combine AST-augmented and code sequence-augmented to represent source code semantic information. We propose an efficient and accurate SCS generation system, Re_Trans. It first utilizes retrieval-based model
to get the most similar code w.r.t semantic and its SCS (S_RM). Then, it feeds the given code and its similar code to the trained discriminator. Finally, it decides to use the S_RM as a result or utilize transformer model to get the new result according to the discriminator’ output. Moreover, we conduct a series of contrast and ablation experiments to demonstrate that the Re_Trans outperforms existing SOTA methods. Combined with the recent work and the research of this paper, we suggest some valuable research points for the future:

In the future, we plan to expand the SCS retrieval library, and pay a special focus on the quality of the expansion data. Furthermore, we also plan to further investigate the usefulness of our approach, using it to generate SCS for other program languages that without code comments. Besides, large-scale language training model will be an inevitable requirement with the increasing daily data. Thereby, it is a meaningful research direction that extracting effective semantic information without occupying too many computing resources.

7 Author Contributions

Conceptualization and Methodology design: Zhang, C.; Data curation, Original draft preparation, Zhang, C. Qiao, M. and Tang, K.; review and editing, Zhou, Q. and Liu F.; Visualization and Investigation, Zhang, C. and Xu, L.

8 Conflict of Interest

The authors have no competing interests to declare that are relevant to the content of this article.

References


International Conference on Software Engineering, Honolulu, HI, USA, 21-28 May 2011.


