Academic Journal of Science and Technology
ISSN: 2771-3032 | Vol. 8, No. 1, 2023

The Source Code Comment Generation Based on Deep Reinforcement Learning and Hierarchical Attention

Daoyang Ming[1], Weicheng Xiong[2, *]

[1]School of Foreign Languages, BaoShan University, Bao Shan, 678000, China
[2]School of Mathematics and Big Data, Guizhou Normal University, Gui Yang, 550018, China
[*] Corresponding author

Abstract: Code summarization provides the main aim described in natural language of the given function, it can benefit many tasks in software engineering. Due to the special grammar and syntax structure of programming languages and various shortcomings of different deep neural networks, the accuracy of existing code summarization approaches is not good enough. This work proposes to adopt the hierarchical attention mechanism to enable the code summarization framework to translate three representations of source code to the hidden spce and then it injects them into a deep reinforcement learning model to enhance the performance of code summarization. We conduct a few of experiments, and the results of which prove that the proposed approaches can obtain better accuracy compared with the baseline approaches.

Keywords: Deep learning, Source Code Comment Generation, Deep Reinforcement Learning, Hierarchical Attention.

1. Introduction

Source code summarization is the task of writing brief natural language descriptions of code [1, 2, 3, 4]. These descriptions have long been the backbone of developer documentation such as JavaDocs [5]. The idea is that a short description allows a programmer to understand what a section of code does and that code's purpose in the overall program, without requiring the programmer to read the code itself. Summaries like "uploads log files to the backup server" or "formats decimal values as scientific notation" can give programmers a clear picture of what code does, saving them time from comprehending the details of that code.

Trace its technological development,at first, the dominant strategy was based on sentence templates and heuristics derived from empirical studies [6-10]. Starting around 2016, data-driven strategies based on neural networks came to the forefront, leveraging gains from both the AI/NLP and mining software repositories research communities [11-14].As far as we know, the existing deep learning based comment generation approaches mainly utilize the seq2seq model in which the program code is encoded into hidden space first and then decode it to produce the target comment.However, these kind of approaches have the following drawbacks: **(1)** they mainly take the source code as plain text and ignore the hierarchical structure of the source code; **(2)** most of the approaches only consider simple features,such as,tokens,which overlooking the hidden information that can help grab the relationships between source code and comments; **(3)** they typically train the decoder to produce the code annotation by calculating and maximizing the odds based on the subsequent natural language words, however in fact, they mainly produce the code annotation from scratch. Therefore, these drawbacks result in inferior comment generation accuracy and inconsistent of the generated comment.

To solve the limitations described above,this work proposes to utilize the hierarchical attention mechanism to combine several source code features for code summarization.The representation of these features are input to the deep reinforcement learning framework for the comment generation task. To present the hierarchical structure of the source code under different contexts considering the code features, the proposed approach allocate weights to different statements and tokens respectively when forming the representation of the source code.Furthermore, the reinforcement learning model improves the generated comment through the actor and critic network, in which given the present state the actor furnishes the confidence of generating the next words, and then the critic network calculates the reward values of the potential next word to provide clues for the generation explorations. At last, We train the framework based on the reward and perform experiments bases on the dataset collected from real projects in github.

2. Related Work

Automatic comment generation approaches vary from manually-crafted templates [26,27, 28], IR [29,30, 31, 32] to neural models [33,34,35].

Comment generation based on manually-craft templates was one of the common methods for generating comments. Sridhara et al. [36] developed the **Software Word Usage Model (SWUM)** to capture the occurrences of terms in source code and their linguistic and structural relationships and then defined different templates for different semantic segments in source code to generate readable natural language. Moreno et al. [37]defined heuristic rules to select relevant information in the source code, and then divided the comments into four parts, and defined different text templates for each part to generate natural language descriptions. McBurney et al. [38] also used the SWUM model to extract the keywords in the Java method, employed the PageRank algorithm to select the important methods in the given method's context, and used a template-based text generation system to generate comments. These frameworks have achieved good results on Java classes and methods.

IR techniques have been widely used in comment generation task. Haiduc et al. [39] used two IR techniques, Vector Space Model and Latent Semantic Indexing, to retrieve relevant terms from a software corpus, and then

organized these terms into comments. Eddy et al. [40] used hierarchical **PAM**, a probabilistic model that selected relevant terms from the corpus and included them to the comments. Unlike the first two research works, Wong et al. [43] proposed that code snippets and their descriptions on the **Q&A** sites can be used to generate comments for a piece of code. They used a token-based code clone detection tool SIM to detect similar code snippets and used their comments as target comments. Wong et al. [42] further thought that the resources of the Q&A sites were limited and proposed to use token-based code clone detection tools to retrieve similar code snippets from GitHub and leverage the information obtained from their comments to generate comments.

Recently many neural networks have been proposed for comment generation. With large-scale corpora for training, neuralbased approaches quickly became state-of-the-art models on this task. Iyer et al. [16] first introduced the seq2seq model from neural machine translation into comment generation, whose encoder is the token embedding and decoder is an **LSTM**. Their model outperforms traditional methods on C# and SQL summaries. Inspired by the difference between natural language and programming language, Hu et al. [15] proposed a neural model named **DeepCom** to capture the structural information of source code. They proposed a structure-based traversal method, using one LSTM to process the AST's traversal sequence, and the other **LSTM** to generate comments for Java methods. LeClair et al. [25] proposed a neural method to predict the comment by combining the sequence information and structure information of the source code with two GRU encoders. In addition, they reconstructed the benchmark dataset for this task, removed duplicate and auto-generated code in the dataset, and divided the dataset into training, validation, and test by project.

This work proposes to adopt the hierarchical attention mechanism to enable the code summarization framework to translate three representations of source code to the hidden spce and then it injects them into a deep reinforcement learning model to enhance the performance of code summarization.

3. The Comment Generation Based on DRL and Hierarchical Attention

Figure 1 presents the overview framework of the proposed comment generation approach through the hierarchical attention mechanism, which mainly follows the actor-critic framework.In specific, four submodules are included in the proposed framework, namely **(a)** the representation module of the source code which is used to explain the structural and unstructured syntax of the code snippet; **(b)** the module of hierarchical attention is utilized to translate the representations of the source code into vectors; **(c)** the LSTM-based text generation module which used to generate the next word according to the words before; and **(d)** the critic module is adopted to estimate the accuracy of the generated annotation and supply feedback the the above modules.

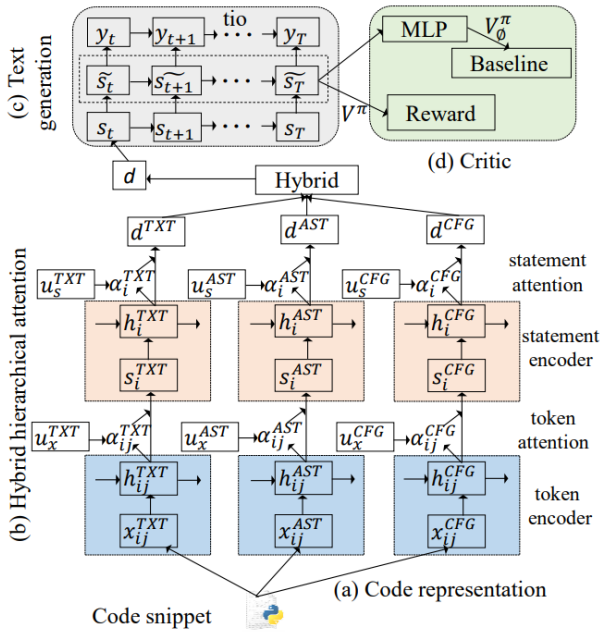


Figure 1. The overview of the proposed code summarization via hierarchical attention mechanism: (a) the three representations of source code are signed as x_{ij}^{TXT} , x_{ij}^{AST} and x_{ij}^{CFT} where *i* and *j* represent the *j*th token in the *i*th statement; (b) d^{TXT} , d^{AST} and d^{CFG} denote the vectors encoded of the three code representations by the hierarchical attention mechanism respectively; (c) the annotation of the source code is produced by the LSTM-based decoder; (d) Given the state $\mathbf{s}_t$, the critic network estimates its value according to|baseline-reward|.

3.1. Representations of the source code

Considering the characteristic of the source code, We adopt a set of symbols, i.e., *{. , " ' _ ( ) { } : ! - (space)* } to tokenize and split the code into different identifiers and then translate them to lowercase letter. Next, the gotten words are embedded into vectors according to the module **genism** in Python. The tokens have not appeared are regarded as unknown words which is similar to [16, 17, 18]. In this chapter, We utilize the following three representations of source code: Plain text sequence, the sequenced abstract syntax tree, and the sequenced control flow graph.

3.1.1. Plain Text representation

Consider that the comments are usually generated by choosing the lexical tokens of the program code, for example the function name, operator name etc., thus one main representation of the source code is the plain text.

3.1.2. Structural representation

When executing the program, the compiler transforms the source code into intermediate code by constructing the abstract syntax tree [19] and then translating to control flow graph which reflects the vital information of the source code. Therefore, the abstract syntax tree and control flow graph are also adopted as the structural representations of the source code. Based on the ast module [20] in Python, the sequenced abstract syntax tree can be obtained. For the control flow graph representation of the source code, each node includes a sequence of tokens which composes the statement and each edge which connects two nodes represents the flow of statements in the source code. Following the ast module [1] and [15], the control flow graph can be obtained and then it is

transformed to get the control flow sequence in depth-first order.

3.1.3. Hybrid hierarchical attention network

Different part of the source code contributes differently to the final output of the comment. In specific, in different code snippet, the same token or statement is deferentially important. Besides, the source code has the hierarchical structure essentially, for example statements are consisted of tokens and the functions are consisted of statements respectively. Thus, We introduce the hierarchical attention mechanism [21], which is proposed in natural language processing, to enable the proposed comment generation approach to pay attention deferentially to different statements and tokens respectively when forming the representations of the source code. As shown in Figure 1 (b), a two-layer attention network are utilized in the proposed approach, they are ate the token layer and the statement layer respectively. The utilized network is consisted of four parts: the token encoder, the token-level attention, the statement encoder, and the statement-level attention. Assumed that the vectors of the the three code representations are represented as d_{TXT} , d_{AST}, and d_{CFG} respectively. Then, they are integrated into one vector d to obtain the final representation of the source code. The detailed information of this network are described as follows.

The Token Level encoding. Assuming that a sequence of tokens $\mathbf{x}_{i0}$,..., $\mathbf{x}_{iTi-1}$ consists a statement s_i , and T_i means the length of the tokens in the statement sequence. Firstly, the tokens are embedded into vectors by the embedding matrix $\mathbf{W}_i$, namely, $\mathbf{v}_{it} = \mathbf{W}_i x_{it}$. Then, as shown in Equation(1), the LSTM model is utilized to encode the tokens from $\mathbf{x}_{i0}$ to $\mathbf{x}_{iTi-1}$ in the statement.

$$\begin{aligned} \mathbf{v}_{it} &= \mathbf{W}_i \mathbf{x}_{it}, t \in [0, T_i) \\ \mathbf{h}_{it} &= lstm(\mathbf{v}_{it}), t \in [0, T_i) \end{aligned} \tag{1}$$

Consider that different tokens in the statement supply different semantic information and contribute differently to the generated comment. For example, in Figure 1 which shows the representation of the source code in the editor, as the words “numbers” and “string” are contained in the given comment, thus the tokens “number” and “str” can supply more information than token “def ” in statement “def check_number_exist(str):” essentially. Therefore, the attention mechanism is adopted by the proposed approach to collect the tokens that are more important for the generating of the comment and the extracted words are merged to generate the representation of the corresponding statement as shown in Equation (2).

$$\begin{aligned} \mathbf{u}_{it} &= tanh(\mathbf{W}_x \mathbf{h}_{it} + \mathbf{b}_x) \\ \alpha_{it} &= \frac{exp(\mathbf{u}_{it}^T \mathbf{u}_x)}{\Sigma_T exp(\mathbf{u}_{it}^T \mathbf{u}_x)} \\ \mathbf{s}_i &= \Sigma_T \alpha_{it} \mathbf{h}_{it} \end{aligned} \tag{2}$$

```
# Check if there are numbers in a string.

def check_number_exist(str):
    has_number = False
    for c in str:
        if c.isnumeric():
            has_number = True
            break
    return has_number
```

Figure 2. The source code example that tokens supply information differently for comment generation.

$\mathbf{W}_x$ represents the weight matrix, $\mathbf{b}_x$ denotes the bias vector, the attention from token $\mathbf{x}_{it}$ to the statement $\mathbf{s}_i$ is signed as $\boldsymbol{\alpha}_{it}$, and $\mathbf{u}_x$ means the sequence vector in the token level. Particularly, $\mathbf{u}_x$ is initialized randomly and optimized during the training process gradually.

The Statement Level Encoding. As the statement vector $\mathbf{s}_i$ has been obtained, similar to the encoding of the tokens, the function vector can be generated. Firstly, the statements are encoded by LSTM according to Equation (3).

$$\mathbf{h}_i = lstm(\mathbf{s}_i), i \in [0, L) \tag{3}$$

where L means the statements number contained in the code snippet. To extract the statements which contain more important information to the corresponding code snippet for the comment generation task, the attention mechanism is adopted again to generate the function vector us in the statement level which enables the framework to estimate the importance of different statements.

$$\begin{aligned} \mathbf{u}_i &= tanh(\mathbf{W}_s \mathbf{h}_i + \mathbf{b}_s) \\ \alpha_i &= \frac{exp(\mathbf{u}_i^T \mathbf{u}_s)}{\Sigma_L exp(\mathbf{u}_i^T \mathbf{u}_s)} \\ \mathbf{d}^c &= \Sigma_L \alpha_i \mathbf{h}_i \end{aligned} \tag{4}$$

Where, the weight matrix is signed as $\mathbf{W}_s$, $\mathbf{b}_s$ represents the bias vector, and the attention from each statement $\mathbf{s}_i$ to the final representation vector d^c is signed $\boldsymbol{\alpha}_i$.

As the vectors of the three representations of the source code, namely the vector of the plain text sequence, the vector of the sequenced abstract syntax tree and the vector of the sequenced control flow graph have been generated, they are concatenated firstly and then are feed into the linear network:$\mathbf{d}=\mathbf{W}_d[\mathbf{d}^{TXT};\mathbf{d}^{AST};\mathbf{d}^{CFG}]+\mathbf{b}_d$, where d denotes the final representation of the source code, $[\mathbf{d}^{TXT};\mathbf{d}^{AST};\mathbf{d}^{CFG}]$ represents the concatenation of the three representations of the source code. At last, an additional hidden layer is utilized for the comment generation: $\mathbf{s}_t = \tanh(\mathbf{W}_c \mathbf{s}_t + \mathbf{b}_d)$, where st is hidden state and $\mathbf{s}_0$ is initialled as $\mathbf{d}$.

3.2. Text generation

As the representation of the source code has been deduced, then comment can be generated by a softmax function. Assume that the policy π defined from the actor network is signed as $\mathbf{p}^{\pi}$, and the distribution of the probability of the tth word y_t is signed as $\mathbf{p}_{\pi}(\mathbf{y}_t | \mathbf{s}_t)$, we can get the comment generation equation:

$$p_{\pi}(y_t | \mathbf{s}_t) = softmax(\mathbf{W}_s \tilde{\mathbf{s}}_t + \mathbf{b}_s) \tag{5}$$

3.3. Critic network

Unlike the traditional code summarization approaches which generate comments by optimizing the probability of next words based on the ground truth, while the comments are produced by optimizing the reward iteratively in the proposed approach which is realized by reinforcement learning. In specific, the critic network is introduced to calculate the reward of the summary action to supply a feedback to train the network iteratively.

4. Experiments and Analysis

To estimate the performance of the proposed approach, sufficient experiments based on real-world dataset is performed. The following research questions are designed for the estimation in the experiment.

RQ1. What is the performance of different components in the proposed model? For instance, whether the representations of the source code, the hierarchical attention network and the reinforcement learning can improve the performance respectively?

RQ2. What is the performance of the proposed code summarization approaches considering different code or comment length?

RQ3. What is the time consumption of different parts?

The research question 1 aims to estimate the performance of each component in the proposed approach compared with the state-of-the-art baselines. The research question 2 aims to estimate the proposed approach consider the range of the code and comment length. The research question 3 is proposed to estimate the time complexity of the proposed approach.

4.1. Dataset preparation

To estimate the performance of the proposed approach, the dataset collected in [14], which is collected from **GitHub** [14] which is the most popular open source projects hosting platform, is utilized and it is a commonly used dataset these years. There are mainly about 108,700 <code, comment> pairs included in the dataset, and about 50,400 code tokens and 31,300 comment tokens are contained respectively in the datset. The dataset is split into different part in a random way, the first part contains 60% for training, the second part contains 20% for validation and the last 20% is used for testing.

4.2. Evaluation metrics

In this work, similar as [8, 100, 113], We estimate the accuracy of the produced annotations from the aspect of their similarity with the corresponding ground-truth comments. In specific, the three widely-used evaluation metrics adopted in NLP area particularly in the the **NMT** task are utilized: **BLEU**[17], **METEOR**[22], and **ROUGE**[18].These metrics mainly calculate the similarity degree between the generated natural language text and the ground-truth by measuring the frequency of the tokens occurrence in both of them from different aspects. As comment generation is also a text generation task in which the natural language words composed the out put, thus they are adopted to assess the accuracy of the produced annotations. Particularly, **BLEU** is the most common evaluate metric adopted in the natural language generation task [22-23] which estimates the **n-gram** accuracy by comparing with a few reference sentences.

Table 1. Performance of different code representations.

Approaches	BLEU-1	BLEU-2	BLEU-3	BLEU-4	METEOR	ROUGE-L
TXT	19.51	2.45	0.95	0.65	5.65	31.56
AST	18.97	3.95	1.87	0.89	5.97	31.23
CFG	19.20	2.45	1.12	0.67	5.12	31.46
TXT&AST	26.56	3.96	1.89	1.32	6.21	37.68
TXT&CFG	27.66	4.25	1.97	1.12	6.38	38.24
AST&CFG	26.35	2.65	0.96	0.97	5.87	38.13
All	**33.16**	**12.39**	6.21	5.10	**9.43**	**46.23**

Table 2. Performance of the hierarchical attention network.

Attention type	BLEU-1	BLEU-2	BLEU-3	BLEU-4	METEOR	ROUGE-L
no atten	18.76	8.21	4.98	3.46	8.24	35.28
1-layer atten	25.79	8.45	5.73	4.67	8.79	38.49
2-layer atten	**33.16**	**12.39**	**6.21**	**5.10**	**9.43**	**46.23**

Table 3. Performance of deep reinforcement learning.

Approach	BLEU-1	BLEU-2	BLEU-3	BLEU-4	METEOR	ROUGE-L
No DRL	26.89	7.21	3.76	2.31	8.21	35.78
With DRL	**33.16**	**12.39**	**6.21**	**5.10**	**9.43**	**46.23**

4.3. Experimental settings

In the experiment, We set vector size of the hidden layers of **LSTM** to be 512 for both encoder and decoder. The mini-batch size is initialized to 32, and the learning rate is initialized to 0.001. Firstly, the the actor and critic network are pretrained by 10 epochs respectively, and then the whole framework is trained by 10 epochs simultaneously. All the experiments are implemented based on Python 3.7.

4.4. RQ1: The performance analysis of different components

4.4.1. The performance considering different code representations

We estimate the proposed approach by different settings considering different components. The three different code representations are signed as **TXT**, **AST** and **CFG** respectively. The hierarchical attention mechanism is denoted as **HAN** and **DRL** refers to deep reinforcement learning.

• **TXT+HAN+DRL**. This baseline regards the source code as plain text and which is encoded by the LSTM-based hierarchical attention network and then train the model by DRL.

• **AST+HAN+DRL.** This approach takes the sequenced abstract syntax tree as the representation of the source code and then encodes it by the **LSTM**-based hierarchical attention network.

•**CFG+HAN+DRL.** This approach utilizes the sequenced control flow as the source code representation and then use the same framework.

• **TXT&AST+HAN+DRL.** In this approach, the plain text sequence and the sequenced abstract syntax tree as the representations of the source code and the framework concatenates their encoded vectors to obtain the hybrid code representation for the comment generation.

• **TXT&CFG+HAN+DRL**. As the same, the plain text sequence and the sequenced control flow are adopted as the representations of the source code and then follows the same framework.

•**AST&CFG+HAN+DRL**.In this approach, the sequenced abstract syntax tree and the sequenced control flow are utilized as the representations of the source code and then follows the same framework.

•**The proposed approach:**

TXT&AST&CFG+HAN+DRL. This is the proposed approach in which the plain text sequence, the sequenced abstract syntax and the sequenced control flow are adopted as the representations of the source code. Then, the LSTM-based hierarchical network is utilized to encode them into vectors and then a hybrid layer is utilized to concatenated them into vector.

The experimental results compared between the proposed approach and the baselines described above is shown in Table

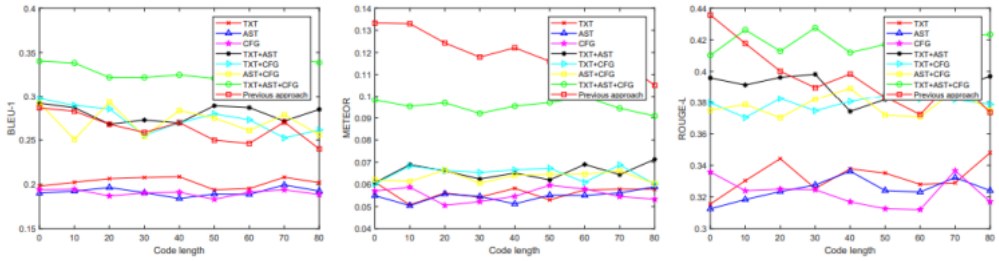
1. From the results we can see that the proposed approach can outperform almost all baselines considering most estimation metrics. In specific, the accuracy of generated comment by the proposed approach can outperform the baselines which use the plain text, the sequenced abstract syntax tree or the sequenced control flow by 29.46% to 31.42% in terms of BLEU-1. Besides, the propose approach improves the generated comments accuracy by 16.58% to 20.53% in terms of BLEU-1 compared with approaches in which two source code representations are adopted. Furthermore, similar result trends are obtained considering the other estimation metrics. To sum up, better performance can be performed by the proposed approach according to the finer-grained code representations for code summarization.

4.4.2. The performance of hierarchical attention mechanism

To estimate the performance of the hierarchical attention mechanism, We design the baselines by encoding the code representations with no attention, with 1-layer attention and with 2-layer attention respectively. In the no attention baseline, the code representations are encoded by normal LSTM without attention. While the code representations of the source code are encoded from tokens to function directly in the 1-layer attention baseline. In the proposed approach, the hierarchical structure of the code representations is adopted and the 2-layer attention mechanism is utilized. The results is presented in Table 3.2, from which wen can observe that the baseline performed by 1-layer attention can achieve better performance by 2.92% to 37.47% consider the estimation metrics. While the proposed approach can improve the baseline with 1-layer attention by 4.36% to 49.59% considering different estimation metrics. These results demonstrate that the adopted hierarchical attention can improve the performance dramatically.

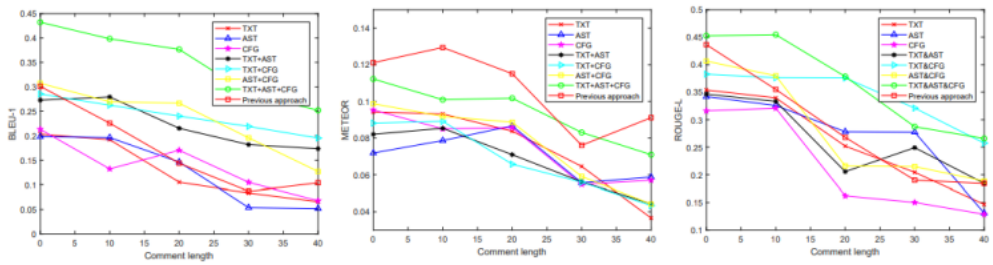
4.4.3. The performance of deep reinforcement learning

To estimate the performance of the proposed reinforcement learning component, We train the model by withing and without the component of reinforcement learning, and they are signed as “approach with DRL” and “approach without DRL”. The corresponding results are shown in Table 3.3, from which we can see that the proposed approach can achieve better values by 14.13% to 130.30% compared with the baseline without reinforcement learning. These results can prove that the proposed approach can improve the comment generation performance significantly.



(a) BLEU-1 (b) METEOR (c) ROUGE-L

Figure 3. The results trend vs. the code length.



(a) BLEU-1 (b) METEOR (c) ROUGE-L

Figure 4. The results trend vs. the comment length.

4.5. RQ2: Performance considering different code and comment length

To measure the influence on the code summarization performance considering different code and comment lengths, We vary the split of the dataset by the lengths of code and comment respectively. Figures 3 and 4 demonstrate the results, and from which we can see that the best performance can be achieved by the proposed approach compared with the baselines from the aspect of all the utilized metrics. From the aspect of BLEU, the proposed approach improves the baselines that adopted different representations of source code by 35.74%, 43.31%, 41.13%, 17.04%, 1l6.97%, and 12.66% respectively when the source code contains 40 tokens. In specific, the baselines that adopted two representations of the source code can all achieve better performance than the baselines which adopts only one representation of the source code. Furthermore, the proposed approach can always improve the performance compared with the baselines which adopted two representations of the source code. For other estimation metrics, the similar results can be achieved. These phenomenons can prove the effectiveness of the proposed code representation component.

Table 4. The time cost to train different models (mins).

	TXT	AST	CFG	TXT&AST	TXT&CFG	AST&CFG	All
Actor	20	24	23	32	31	33	39
Critic	27	30	29	41	40	41	50
A2C	36	41	43	50	51	53	58

The code summarization result considering different comment lengths is presented in Figure 3.10, from which we can find that the proposed approach performs better compared with the baselines considering different lengths of comment. In specific, when the length of the comment is 20, the performance of the proposed approach improves 107.61%, 100.31%, 200.59%, 51.77%, 42.64%, and 47.77% respectively. We can observe that the performance tends to be worse when the length of the comment increased which is similar to the results in the neural machine translation task [24-25].

4.6. RQ3: Time consumption analysis

To estimate the time complexity of the proposed approach, the average training time of each epoch considering different representations of the source code are recorded. The result is given in Table 4, from which we can find that for all the three stages in the proposed approach, each epoch costs less than 1 hour for training. This result shows that the time complexity of the proposed approach is low and the approach is reasonable for practical usage.

4.7. Case study

We demonstrate four real-world code examples for generating their comments using our approach in Table 3.5. In this table, we first show the code snippet in the second line and then give the ground truth comment which is the code comment that is collected together with the code snippet from GitHub. Next, the generated comments by different approaches are given.For our approach, shown as 2-layer+DRL, we have highlighted the words that are closer to the ground truth. It can be observed that the generated comments by our approach are the closest to the ground truth. Although the approaches with **DRL (1-layer+DRL)** can generate some tokens which are also in the ground truth, they cannot predict those tokens which do not frequently appear in the training data, i.e., object in the case example.On the contrary, the deep-reinforcement-learning-based approach can generate some tokens which are closer to the ground truth, such as process. This can be illustrated by the fact that our approach has a more comprehensive exploration on the word space and optimizes the BLEU score directly.

Table 5. Case study of code summary generated by each approach.

		Case example

Code snippet		`def Pool(processes=None, initializer=None, initargs=(), maxtasksperchild=None):`
		`from multiprocessing.pool import Pool`
		`return Pool(processes, initializer, initargs, maxtasksperchild)`

Ground truth		returns a process pool object.
Generated Comments	no attention	returns a list of all available vm sizes on the cloud provider.
	1-layer	returns the total number of cpus in the system.
	2-layer	return a list of all elements in te given order.
	1-layer+DRL	returns a process object with the given id.
	2-layer+DRL	returns a process object.

5. Threats to Validity

The threat to validity is that we evaluate the proposed approach only based on the dataset which is consisted of Python code and comment pairs, therefore it may be unrepresentative of code summarization considering other programming languages. However, as the components in the proposed approach are all general models which can be adopted to realize the code annotation regarding other programming language. Besides, this approach is based on static analysis which could be a barrier to adopt the proposed approach, for example, to obtain the effective static analysis results, significant effort should be made.

6. Conclusion

In this work, We propose to utilize three representations of the source code considering both the unstructured and structural features, namely, the plain text, the sequenced abstract syntax tree and the sequenced control flow to represent the hierarchical structure of the source code. And the two-layer hierarchical attention mechanism is adopted to encode the source code representations.To estimate the effectiveness of the proposed approach, a few of experiments based on the dataset grabbed from real world projects are performed. The results prove the high quality of the generated comment.

Acknowledgment

The authors gratefully acknowledge the financial support from science and technology planning project of Yunnan Province (Joint project of local universities) (202001BA070001-096) and Fund Project of Yunnan Provincial Education Department.

References

[1] N.KALCHBRENNER,L. ESPEHOLT, K. SIMONYAN, A. V. D. OORD, AND K. KAVUKCUOGLU, Neural machine translation in linear time, arXiv preprint arXiv:1610.10099, (2016).

[2] A.GRAVES,Generating sequences with recurrent neural networks, arXiv preprint arXiv:1308.0850, (2013).

[3] A.LOUIS,S.K. DASH, E. T. BARR, AND C. SUTTON, Deep learning to detect redundant method comments, arXiv preprint arXiv:1806.04616, (2018).

[4] R.COLLOBERT AND J.WESTON, A unified architecture for natural language processing: Deep neural networks with multitask learning, in Machine Learning, Proceedings of the Twenty-Fifth International Conference (ICML 2008), 2008, pp. 160–167.

[5] S.IYER,I.KONSTAS,A.CHEUNG,ANDL.ZETTLEMOYER, Summarizing source code using a neural attention model., in Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (1), 2016, pp. 2073–2083.

[6] M. MALHOTRA AND J. K. CHHABRA, Class level code summarization based on dependencies and micro patterns, in 2018 Second International Conference on Inventive Communication and Computational Technologies (ICICCT), IEEE, 2018, pp. 1011–1016.

[7] A. SEE, P. J. LIU, AND C. D. MANNING, Get to the point: Summarization with pointer-generator networks, in Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), Vancouver,Canada,July 2017, Association for Computational Linguistics, pp. 1073–1083.

[8] H.SHI, H. ZHOU, J. CHEN, AND L. LI, On tree-based neural sentence modeling, arXiv preprint arXiv:1808.09644, (2018).

[9] K.PAPINENI, S.ROUKOS,T. WARD, AND W. J. ZHU, Bleu: a method for automatic evaluation of machine translation, in Proceedings of the 40th annual meeting on association for computational linguistics, Association for Computational Linguistics, 2002, pp. 311–318.K.

[10] M.HAMMAD,A.ABULJADAYEL,ANDM.KHALAF, Summarizing services of java packages, Lecture Notes on Software Engineering, 4 (2016), pp. 129–132.

[11] L. MORENO, J. APONTE, G. SRIDHARA, A. MARCUS, L. POLLOCK, AND K. VIJAYSHANKER, Automatic generation of natural language summaries for java classes, in 2013 21st International Conference on Program Comprehension (ICPC), IEEE, 2013, pp. 23–32.

[12] R. COLLOBERT AND J. WESTON, A unified architecture for natural language processing: Deep neural networks with multitask learning, in Machine Learning, Proceedings of the Twenty-Fifth International Conference (ICML 200

[13] N. KALCHBRENNER, L. ESPEHOLT, K. SIMONYAN, A. V. D. OORD, AND K. KAVUKCUOGLU, Neural machine translation in linear time, arXiv preprint arXiv:1610.10099, (2016).8), 2008, pp. 160–167.

[14] N. J. ABID, N. DRAGAN, M. L. COLLARD, AND J. I. MALETIC, Using stereotypes in the automatic generation of natural language summaries for c++ methods, in 2015 IEEE International Conference on Software Maintenance and Evolution (ICSME), IEEE, 2015, pp. 561–565.

[15] X. Hu, G. Li, X. Xia, D. Lo, and Z. Jin. Deep code comment generation. In Proceedings of the 26th International Conference on Program Comprehension, pages 200–210. ACM, 2018.

[16] Github, https://github.com/.

[17] K. PAPINENI, S. ROUKOS, T. WARD, AND W. J. ZHU, Bleu: a method for automatic evaluation of machine translation, in Proceedings of the 40th annual meeting on association for computational linguistics, Association for Computational Linguistics, 2002, pp. 311–318.

[18] V. R. KONDA AND J. N. TSITSIKLIS, Actor-critic algorithms, in Advances in neural information processing systems, 2000, pp. 1008–1014.

[19] E. M. PONTI, R. REICHART, A. KORHONEN, AND I. VULIC, Isomorphic transfer of syntactic structures in cross-lingual nlp, in The 56th Annual Meeting of the Association for Computational Linguistics, 2018, pp. 1531–1542.

[20] J. POUGET-ABADIE, D. BAHDANAU, B. V. MERRIENBOER, K. CHO, AND Y. BENGIO, Overcoming the curse of sentence length for neural machine translation using automatic segmentation, arXiv preprint arXiv:1409.1257, (2014).

[21] H. SHI, H. ZHOU, J. CHEN, AND L. LI, On tree-based neural sentence modeling, arXiv preprint arXiv:1808.09644, (2018).

[22] 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, vol. 29, 2005, pp. 65–72.

[23] Z. YANG, D. YANG, C. DYER, X. HE, A. SMOLA, AND E. HOVY, Hierarchical attention networks for document classification, in Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL, 2016, pp. 1480–1489.

[24] M. ALLAMANIS, H. PENG, AND C. SUTTON, A convolutional attention network for extreme summarization of source code, in International Conference on Machine Learning, 2016, pp. 2091–2100.

[25] 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, in Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering, ACM, 2018, pp. 397–407.

[26] Paul W. McBurney and Collin McMillan. 2016. Automatic Source Code Summarization of Context for Java Methods. IEEE Trans. Software Eng. 42, 2 (2016), 103–119. https://doi.org/10.1109/TSE.2015.2465386

[27] Laura Moreno, Jairo Aponte, Giriprasad Sridhara, Andrian Marcus, Lori L. Pollock, and K. Vijay-Shanker. 2013. Automatic generation of natural language summaries for Java classes. In IEEE 21st International Conference on Program Comprehension, ICPC 2013, San Francisco, CA, USA, 20-21 May, 2013. 23–32. https://doi.org/10. 1109/ICPC.2013.6613830.

[28] Giriprasad Sridhara, Emily Hill, Divya Muppaneni, Lori L. Pollock, and K. VijayShanker. 2010. Towards automatically generating summary comments for Java methods. In ASE. ACM, 43–52.

[29] Brian P. Eddy, Jeffrey A. Robinson, Nicholas A. Kraft, and Jeffrey C. Carver. 2013. Evaluating source code summarization techniques: Replication and expansion. In IEEE 21st International Conference on Program Comprehension, ICPC 2013, San Francisco, CA, USA, 20-21 May, 2013. 13–22. https://doi.org/10.1109/ICPC.2013. 6613829.

[30] Sonia Haiduc, Jairo Aponte, Laura Moreno, and Andrian Marcus. 2010. On the Use of Automated Text Summarization Techniques for Summarizing Source Code. In 17th Working Conference on Reverse Engineering, WCRE 2010, 13-16 October 2010, Beverly, MA, USA. 35–44. https://doi.org/10.1109/WCRE.2010.13

[31] Martin White, Michele Tufano, Christopher Vendome, and Denys Poshyvanyk. 2016. Deep learning code fragments for code clone detection. In Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering, ASE 2016, Singapore, September 3-7, 2016. 87–98. https://doi.org/10.1145/2970276. 2970326.

[32] Edmund Wong, Taiyue Liu, and Lin Tan. 2015. CloCom: Mining existing source code for automatic comment generation. In 22nd IEEE International Conference on Software Analysis, Evolution, and Reengineering, SANER 2015, Montreal, QC, Canada, March 2-6, 2015. 380–389. https://doi.org/10.1109/SANER.2015.7081848.

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

[34] Srinivasan Iyer, Ioannis Konstas, Alvin Cheung, and Luke Zettlemoyer. 2016. Summarizing Source Code using a Neural Attention Model. In Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016, August 7-12, 2016, Berlin, Germany, Volume 1: Long Papers. https://www.aclweb. org/anthology/P16-1195/

[35] Alexander LeClair, Siyuan Jiang, and Collin McMillan. 2019. A neural model for generating natural language summaries of program subroutines. In Proceedings of the 41st International Conference on Software Engineering, ICSE 2019, Montreal, QC, Canada, May 25-31, 2019. 795–806. https://doi.org/10.1109/ICSE.2019.00087.

[36] Giriprasad Sridhara, Emily Hill, Divya Muppaneni, Lori L. Pollock, and K. VijayShanker. 2010. Towards automatically generating summary comments for Java methods. In ASE. ACM, 43–52.

[37] Laura Moreno, Jairo Aponte, Giriprasad Sridhara, Andrian Marcus, Lori L. Pollock, and K. Vijay-Shanker. 2013. Automatic generation of natural language summaries for Java classes. In IEEE 21st International Conference on Program

Comprehension, ICPC 2013, San Francisco, CA, USA, 20-21 May, 2013. 23–32. https://doi.org/10. 1109/ICPC.2013.6613830.

[38] Paul W. McBurney and Collin McMillan. 2016. Automatic Source Code Summarization of Context for Java Methods. IEEE Trans. Software Eng. 42, 2 (2016), 103–119. https://doi.org/10.1109/TSE.2015.2465386.

[39] Sonia Haiduc, Jairo Aponte, Laura Moreno, and Andrian Marcus. 2010. On the Use of Automated Text Summarization Techniques for Summarizing Source Code. In 17th Working Conference on Reverse Engineering, WCRE 2010, 13-16 October 2010, Beverly, MA, USA. 35–44. https://doi.org/10.1109/WCRE.2010.13.

[40] Edmund Wong, Jinqiu Yang, and Lin Tan. 2013. AutoComment: Mining question and answer sites for automatic comment generation. In 2013 28th IEEE/ACM International Conference on Automated Software Engineering, ASE 2013, Silicon Valley, CA, USA, November 11-15, 2013. 55–67. https://doi.org/10.1109/ASE. 2013.6693113.