IJARSCT
ISSN (Online) 2581-9429
International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
Impact Factor: 7.53
Volume 4, Issue 8, April 2024

# Modified Approach to Code Summarization

**Irene Mariam Sajan[1], Irin Sunny Thekkekara[2], Karishma Anna Koshy[3], Maria Rojo Jose[4], Dr. Varghese Chooralil[5]**
Department of Computer Science and Engineering[1,2,3,4,5]
Rajagiri School of Engineering and Technology, Kochi, Kerala, India

**Abstract:** *Code summarization is the process of automatically generating a concise and informative summary of a given code snippet. It aims to assist software developers in understanding the functionality of the code. In this paper, we review three approaches for code summarization. The approach leverages the hierarchical structure of the abstract syntax tree (AST) of the code, which represents the syntactic structure of the code. The AST is encoded using a tree-LSTM network, which is capable of capturing the structural information of the AST. The encoded tree-LSTM representation of the AST is then fed into a decoder network to generate the summary. The Tree-LSTM-based approach for code summarization has shown promising results in capturing the hierarchical structure of the code and generating accurate and concise summaries. Further research can explore the integration of additional semantic information and the extension of our approach to other programming languages*

**Keywords:** BASTS, AST, Encoder-Decoder, Sentence-Weight algorithm

## I. INTRODUCTION

The field of natural language processing has seen significant advancements in recent years, particularly in the area of generating natural language descriptions of source code. In this overview, we will look at three research papers that propose novel approaches for generating natural language descriptions of source code.

The first paper, “Improved code summarization using blockwise abstract syntax tree splitting” , proposes a new approach to code summarization that uses block-wise abstract syntax tree splitting to divide a complex code block into smaller sub-blocks that can be more easily summarized. The approach involves first splitting the code block into a set of smaller blocks based on the abstract syntax tree of the code. Then, each sub-block is summarized individually using an attention-based neural network that generates a summary for each sub-block. Finally, the individual summaries are combined to generate a summary for the entire code block.

The second paper, "A Multi-Module Based Method for Generating Natural Language Descriptions of Code Fragments", proposes a three-module approach to generate human-like descriptions of code fragments. The approach involves pre-processing the code, internal processing to derive an abstract representation of the code, and external processing to generate a description using a sentence weight algorithm and sentence similarity checking. The evaluation showed that the proposed approach outperforms existing methods in terms of accuracy and fluency.

The third paper, "Source Code Summarization Using Attention-based Keyword Memory Networks", proposes an approach that uses attention-based keyword memory networks to generate a summary of source code. The proposed approach uses attention mechanisms to identify the most relevant keywords, which are then stored in a keyword memory module. The approach was evaluated on two different datasets, and the results showed that it outperformed existing state-of-the-art methods in terms of accuracy and fluency.

Overall, these three papers highlight the significant progress made in the field of natural language processing for generating natural language descriptions of source code.

These approaches have the potential to improve software documentation, aid in code comprehension, and assist in software maintenance and development.

IJARSCT
ISSN (Online) 2581-9429
International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
Impact Factor: 7.53
Volume 4, Issue 8, April 2024

## II. OVERVIEW

### A. Improved code summarization using blockwise abstract syntax tree splitting

Blockwise abstract syntax tree (AST) splitting is a technique used in code summarization to improve the quality of the generated summaries. The block wise AST splitting approach involves dividing the code into smaller blocks based on its AST structure. The AST is a tree-like representation of the code that shows the syntactic structure of the program. By splitting the code into smaller blocks based on its AST structure, the summarization model can better understand the context and meaning of each block, leading to more accurate and informative summaries.

The blockwise AST splitting technique works by recursively traversing the AST of the code and splitting it into smaller blocks at each node. Each block is then summarized separately, and the summaries are combined to form the final summary of the code.

### B. A Multi-Module Based Method for Generating Natural Language Descriptions of Code Fragments

A Multi-Module Based Method for Generating Natural Language Descriptions of Code Fragments is a research paper that presents a novel approach to generate human-like descriptions of code fragments. The approach consists of three main modules: pre-processing, internal processing, and external processing

The pre-processing module is responsible for parsing the code and extracting relevant information such as variable names, function names, and their corresponding types. This information is then used by the internal processing module to generate an abstract representation of the code fragment. The internal processing module uses a combination of syntactic and semantic analysis techniques to derive the abstract representation.

The abstract representation is then passed to the external processing module, which includes a sentence weight algorithm and sentence similarity checking. The sentence weight algorithm assigns weights to candidate sentences based on their relevance to the code fragment and their fluency. The candidate sentences are then ranked based on their weights, and the top-ranked sentence is selected as the final description of the code fragment. The sentence similarity checking is performed by comparing the generated sentence with a set of reference sentences, which are manually created by human annotators. The similarity is measured using a combination of semantic and syntactic similarity metrics.

### C. Source Code Summarization Using Attention-based Keyword Memory Networks

Source Code Summarization Using Attention-based Keyword Memory Networks is a research paper that proposes a new approach to automatically generate a summary of source code. The proposed approach uses attention-based keyword memory networks, which are neural networks that are capable of selecting the most relevant keywords from the source code to generate a summary.

The approach consists of three main stages: preprocessing, encoding, and decoding. In the preprocessing stage, the source code is tokenized and preprocessed to remove any irrelevant information. In the encoding stage, the tokenized source code is fed into a neural network that uses attention mechanisms to identify the most relevant keywords. These keywords are then stored in a keyword memory module. Finally, in the decoding stage, the keyword memory module is used to generate a summary of the source code.

## III. METHOD -1 USING BLOCK-WISE ABSTRACT SYNTAX TREE SPLITTING[1]

Code summarization is the process of creating a summary when a source code is given as the input. Traditional code summarization methods usually rely on recurrent neural networks(RNN). But these methods cannot handle the complexity of the code and usually do not capture the syntactic and structural information. It is important to understand each token and syntax while summarizing the code. Either NLP syntax trees[2] or AST trees can be used for representing the code. The issue with the NLP syntax tree is that it cannot represent all the details of the code. That is the tree contains a vague idea of the code. Applying a machine learning model on an NLP syntax tree[2] can generate inadequate and meaningless results.To overcome this the source code is initially converted into Abstract Syntax Tree(AST).

An AST is the structural representation of the source code. Usually each of the nodes represents a statement in the code.The AST representation is detailed and complex. Encoding this tree gives better and accurate results.

IJARSCT
ISSN (Online) 2581-9429
International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
Impact Factor: 7.53
Volume 4, Issue 8, April 2024

**Methodology**

Encoding an serialized AST can be implemented using LSTM[3] or GRU. While encoding serialized AST, linear structure is captured but the hierarchical structure cannot be comprehended.

The most relevant work discussed is creating smaller AST's. Encoding these AST using a machine learning model while learning about local and long term dependencies.

**Steps**

- Code Splitting:-Code splitting is done to improve the summary produced. The code is first converted to control flow graphs. This graph shows the flow of control from one node to another. Then this graph is converted to a dominator tree. It is created in such a way that the previous node dominates the successive node and all the paths from first to other nodes pass through it.The edges are removed on the basis of the number of incoming edges. If the count of incoming edges is more then a split is created. The dominator tree is split on the basis of the idea that each block contains more logic than a single statement.
- Tree Encoding:-Creating a vector representation using syntax embedding is the first step. The vector representation is pre-trained and fine tuned before applying a machine learning model.Child -Sum LSTM[6] is applied to this representation. As it processes each node, new hidden states are processed.The values of the input, output and forget gate are also updated at each stage through the process. A sigmoid activation function is used to obtain the result between zero and one. The values closer to zero are not that important and values closer to one are given more priority.
- Summarizing Code With Transformer[5]:-The final step is summarization. It is done using an encoder and decoder model. The embedding of the AST is passed through a pooling layer. Average pooling is preferred in this case.The syntax embeddings and code token embeddings are combined in a feed-forward layer before it is further processed.
- Encoder- It takes the input token by token and encodes the new hidden state information. This step is crucial for the accuracy of the summary.
- Decoder- the decoder is responsible for generating a comment word based on the previous output and the current encodings.

**Benefits**

- The splitting of the AST into blocks is done on the basis that each block can retain sufficient data. BASTS is effective and potent and thereby used in big projects.
- BASTS saves money on grammar notation correction by using training data first, followed by field names, as opposed to starting from scratch with two representations.
- BASTS is helpful as it can understand both long-term dependencies between tokens and local hierarchical connections between grammars.

**Conclusion**

The summaries generated by the proposed approach also demonstrate a better understanding of the code's functionality, capturing the fine-grained structural information of the code.The hybrid architecture of CNNs and transformers further improves the model's ability to capture both local and global dependencies.

## IV. METHOD - 2 MULTI-MODULE BASED APPROACH

Choosing the most important functionality from source code is one of the most critical elements in source code summarization. Logic and conciseness has to be maintained for producing an apt summary for the program. The information presented in a concise format must have the complete idea flow of the program as well. A multi-module based approach is discussed here which eases and presents the process of source code summarization in a clear way.

The three modules are Preprocessing module, internal processing and external processing module. A sentence weight algorithm is then used at the end to present the summary in an optimized manner.

IJARSCT
ISSN (Online) 2581-9429
International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
Impact Factor: 7.53
Volume 4, Issue 8, April 2024

**Methodology**

**Preprocessing module**

This module is mainly used to categorize the given source code into different classifications according to their type. This is mainly done as different type of statements are assigned different priorities.

There are 3 main classifications:

1. Direct statement: These highlight the main behavior of the program. These include statements like type declarations, method calls, assignment of values to variables.
   Example: int x=0;
2. Indirect statement: They serve as a helping hand for direct statements and add on to explaining the points discussed in the code fragment. A program has a huge number of indirect statements. Since they form a majority, they contain a lot of high priority information which aids in developing a summary. They usually contain operation, auxiliary parameters, method calls, variable types and more.
3. Special statement: They mostly contain print statements which contain and convey more sensitive information. The information printed can easily be used to develop summaries.

Example: Consider 2 statements, first one is printf("The sum of the array members is %d",sum of array); and the second is, s=s+c;. It is easier to extract summary from the first sentence rather than the second one, so special statements are always given the highest priority in determining the summary of a code fragment.

**Internal Processing Module**

Here Software Word Usage Model along with CamelCase to find out the characteristics and then natural language descriptions are generated.

The template is of the form: Verb A preposition B A and B get values from the program and verb and
proposition positions are fixed irrespective of the type of
program. 2 types of cases can arise. One is variable assignment type of statements like x = 10. 10 will be the subject and = is taken as an assignment. So the description will be formed as "Add 10 assignment to x".Another is class object instantiation which is printed as "instantiation for class class_name".

**External processing module**

The generated natural language descriptions try to connect with each other to present a logic flow. First the sentences are sorted according to their sentence type weight value. The similarity between sentences is checked. For this, cosine similarity is used.

**Sentence Weight Algorithm**

Sorting is done according to prioritization which is given during the preprocessing module. The weight is taken as the sum of weight value of the sentence and the weight value of that particular subject presented in the sentence.

Sentence Similarity Checking

Cosine similarity is used by first mapping to vectors. Then sentence ordering is done as follows: Let S1 and S2 be sentence types of sentence 1 and sentence 2.

If S1>S2, then S1->S2 If S2>S1, then S2->S1

If both are equal, then sorted according to subject weight.

**Benefits**

Three aspects are taken into consideration while evaluating the above proposed method. They are: accuracy, simplicity and smoothness. The accuracy test was taken by comparing the method with manual methods like TextRank and Suncode. These methods only showed 70% inclusion of subject words while the accuracy rate is 96% for the method discussed above. The simplicity rate is 40% which is less compared to the research work done by various other researchers so this section can be worked upon as a future work. Human evaluators were chosen to judge the summary and showed great results.

IJARSCT
ISSN (Online) 2581-9429
International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
Impact Factor: 7.53
Volume 4, Issue 8, April 2024

**Threats**

While performing internal processing, certain important keywords or information can be lost which can lead to loss of important information. The method was only tested for 2 publicly available datasets. More testing has to be done to ensure that the accuracy rate remains same for all kinds of datasets.

## V. METHOD -3 SOURCE CODE SUMMARIZATION USING ATTENTION-BASED KEYWORD MEMORY NETWORKS [13]

Source code is written in coding languages which gives instructions to computers to perform certain computations. However the description of source code is written in natural language. There is obviously a huge difference in the syntax of natural language and source code, it is very difficult to translate descriptions of source code directly. A two-phase model is created that will help build a bridge between the gap of natural language and its description.

**Methodology**

A two-phase model that contains a keyword predictor and a description generator is created to make effective summaries of source codes. The keyword predictor would identify the most relevant keywords in the code this is done using convolutional neural network models and the description generator uses convolutional neural networks and recurrent neural networks along with the keywords that were earlier predicted to create summaries. Using keywords to generate a summary is easier than without as keywords take the most important parts of the code.

**Steps**

- The process of predicting keywords in source code involves analyzing the code's description to identify nouns and verbs that are relevant to the code's role and operation. This is done by extracting the nouns and verbs from the description and using them as keywords for the corresponding source code.
- To train the model, the extracted keywords are used as examples of what to look for in future code descriptions. The model is then trained to predict which nouns and verbs are likely to appear in the description of a given source code to create accurate summaries. Cross entropy is used to minimize the difference between the target probability distribution and the predicted probability distribution of the keyword prediction layer. [14,15]
- The description generator contains three parts which include the encoder, memory and the decoder. A description generator is a natural language generation model that tries to create an understandable summary of code.
- The encoder converts the source code into a context vector, this represents the main characteristics of the code. This vector representation takes the main features and patterns from the code that are needed to create a summary. To capture important parts of the code, where attention should be given, an attention mechanism is added to the CNNs in the encoder.[16] An attention mechanism focuses on important parts of the code that are needed to create summaries. It assigns weights to the tokens in the input code based on its importance in the summary that is currently being generated.
- The memory contains the keyword memory with an attention mechanism. The keywords that were predicted from the source code can be found here and is retrieved according to its relevance and is given as an input to the decoder.
- The decoder creates a summary of the source code since. As it is creating a summary the previous words have a connection to the current words so LSTM [17] is used. LSTM is a type of recurrent neural network, a node will get an input that will be computed and then an output will be created which will be fed as an input in the next step along with that step's input. Hence it will remember previous inputs which help create logical summaries.

**Benefits**

Some benefits of using keyword memory include:

IJARSCT
ISSN (Online) 2581-9429
International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
Impact Factor: 7.53
Volume 4, Issue 8, April 2024

- Accuracy- Important parts of the code are being detected and used in the summary.
- Speed- It allows faster detection of relevant parts of the code.
- More understandable- It will focus on necessary parts of the code so that beginners can know where to focus their attention in understanding the program.

**Conclusion**

- After experiments were conducted it was shown that more relevant and accurate summaries were created
- when using keywords along with attention as it considered the previous word sequence. While the results without using keywords were not as accurate.
- Keyword memory is very useful in creating accurate summaries and can help reduce the significant difference between source code and its description.

## VI. RESULTS AND DISCUSSION

| Methods | Source Code - Representation | Rouge-1 |
|---|---|---|
| BASTS | SPLIT AST | 0.24 |
| Seq2Seq | (API,comments)(API,code,comments) | 0.3 |
| Transformer | comments, code, AST | 0.20 |
| SWUM and CamelCase | Code statements | 0.24 to 0.34 |
| Attention-based Keyword Memory Networks | Tokens-keywords | 0.35 |

Fig.1. Rouge -1 measures of different code summarization methods

ROUGE stands for Recall-Oriented Baseline Study for Substantive Evaluation. Basically, it is a set of metrics that can be used to evaluate both automatic text summarization and machine translations. It works by comparing an automatically generated summary or translation to a set of reference summaries (usually human-generated)

Rouge-N — This provides information about how unigrams, bigrams, trigrams and higher n-grams overlap.

Rouge-L uses LCS to determine the longest matching word sequence. The advantage of using LCS is that it does not require consecutive matches, but rather consecutive matches that reflect sentence-level word order.

Rouge-S - All pairs of words in a sentence that allow arbitrary spaces are considered ordered.

Here, rouge-1 is a measure which considers unigrams between human-made and system-made summaries. It measures the overlap of these two.

## VII. CONCLUSION

After detailed review and analysis we have concluded that summarizing code using BASTS and Multiway-LSTM provides accurate results. Focus is given to BASTS as it can be used to identify and accurately split AST while maintaining both local and hierarchical dependencies. Multi-way LSTM is used to process data that has multiple relationships between the elements, such as data in a recommendation system or a knowledge graph. Better handling of long-term dependencies: Like regular LSTMs, Multiway-LSTMs are able to capture long-term dependencies in data, which is useful for tasks such as language translation and text generation.

## REFERENCES

[1] Chen Lin, Zhichao Ouyang, Junqing Zhuang, Jianqiang Chen, Hui Li, Rongxin "Improving Code Summarization with Block-wise Abstract Syntax Tree Splitting",29th International Conference on Program Comprehension (ICPC),IEEE/ACM;2021.

[2] X. Hu, G. Li, X. Xia, D. Lo, and Z. Jin, "Deep code comment generation," in ICPC, 2018, pp. 200–210.

IJARSCT
ISSN (Online) 2581-9429
International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
Impact Factor: 7.53
Volume 4, Issue 8, April 2024

[3] X. Zhu, P. Sobhani, and H. Guo, "Long short-term memory over recursive structures," in ICML, vol. 37, 2015, pp. 1604–1612.
[4] S. Hochreiter and J. Schmidhuber, "Long short-term memory," Neural Computation, vol. 9, no. 8, pp. 1735–1780, 1997.
[5] W. U. Ahmad, S. Chakraborty, B. Ray, and K. Chang, "A transformer- based approach for source code summarization," in ACL, 2020, pp. 4998–5007.
[6] Yusuke Shido, Yasuaki Kobayashi, Akihiro Yamamoto, Atsushi Miyamoto,"Automatic Source Code Summarization with Extended Tree-LSTM", 2019 International Joint Conference on Neural Networks (IJCNN); 2019.
[7] Xuejian Gao , Xue Jiang , Qiong Wu , Xiao Wang , Lei Lyu , And Chen Lyu, "A Multi-Module Based Method For Generating Natural Language Descriptions Of Code Fragments",Ieee Access (2021), Doi: 10.1109/Access.2021.3055955
[8] S. Haiduc, J. Aponte, L. Moreno, and A. Marcus, ''On the use of automated text summarization techniques for summarizing source code,'' in Proc. 17th Work. Conf. Reverse Eng., Oct. 2010, pp. 35–44.
[9] Immanuel, R. R., & Sangeetha, S. (2023). Identifying Different Emotions Of Human Using Eeg Signals Using Deep Learning Techniques. Journal Of Theoretical And Applied Information Technology, 101(18).
[10] B. P. Eddy, J. A. Robinson, N. A. Kraft, and J. C. Carver, ''Evaluating source code summarization techniques: Replication and expansion,''in Proc. 21st Int. Conf. Program Comprehension (ICPC), May 2013, pp. 13–22.
[11] L. Moreno, J. Aponte, G. Sridhara, A. Marcus, L. Pollock, and K. Vijay-Shanker, ''Automatic generation of natural language summaries for java classes,'' in Proc. 21st Int. Conf. Program Comprehension (ICPC), May 2013, pp. 23–32.
[12] A. Tuan Nguyen and T. N. Nguyen, ''Automatic categorization with deep neural network for open-source java projects,'' in Proc. IEEE/ACM 39th Int. Conf. Softw. Eng. Companion (ICSE-C), May 2017, pp. 164–166.
[13] S. Iyer, I. Konstas, A. Cheung, and L. Zettlemoyer, ''Summarizing source code using a neural attention model,'' in Proc. 54th Annu. Meeting Assoc. Comput. Linguistics (Long Papers), vol. 1, 2016, pp. 2073–2083.
[14] Immanuel, R. R., & Sangeetha, S. K. B. (2022, August). Analysis of EEG Signal with Feature and Feature Extraction Techniques for Emotion Recognition Using Deep Learning Techniques. In International Conference on Computational Intelligence and Data Engineering (pp. 141-154). Singapore: Springer Nature Singapore.
[15] YunSeok Choi, Suah Kim, Jee-Hyong Lee*, "Source Code Summarization Using Attention-based Keyword Memory Networks" 2020 IEEE International Conference on Big Data and Smart Computing (BigComp)
[16] D. Bahdanau, K Cho, and Y Bengio. Neural machine translation by jointly learning to align and translate. In Proceedings of Interna-tional Conference on Learning Representations, 2015.
[17] M. Denil, A. Demiraj, N. Kalchbrenner, P. Blunsom, and N. de Freitas. Modelling, visualising and summarising docu-ments with a single convolutional neural network. arXiv pre-print arXiv:1406.3830, 2014.
[18] M. Allamanis, H. Peng, and C. Sutton. A convolutional attention network for extreme summarization of source code. In Proceedings of the 33rd International Conference on Machine Learning, pages 2091-2100, 2016.
[19] S. Hochreiter, and J. Schmidhuber. Long short-term memory. Neural computation, 9(8):1735–1780, 1997.