

# **AUTOMATIC COMMENT GENERATION USING DEEP LEARNING TECHNIQUE**

A DISSERTATION

SUBMITTED IN PARTIAL FULFILLMENT OF  
THE REQUIREMENTS FOR THE AWARD OF DEGREE

OF

**MASTER OF TECHNOLOGY**

IN

**SOFTWARE ENGINEERING**

Submitted By

**SARANSH SHARMA**

**(2K21/SWE/22)**

Under the supervision of

**Ms. PRIYA SINGH**

**ASSISTANT PROFESSOR**



**DEPARTMENT OF SOFTWARE ENGINEERING**

**DELHI TECHNOLOGICAL UNIVERSITY**

(Formerly Delhi College of Engineering)

Bawana Road, Delhi - 110042

MAY 2023

M.Tech (Software Engineering)

Saransh Sharma

2023

**DELHI TECHNOLOGICAL UNIVERSITY**

(Formerly Delhi College of Engineering)

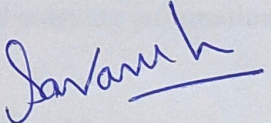
Bawana Road, Delhi-110042

**CANDIDATE'S DECLARATION**

I, **Saransh Sharma**, Roll No – **2K21/SWE/22** student of MTech (Software Engineering), hereby declare that the Project Dissertation titled “**Automatic Comment Generation using Deep Learning Technique**” which is submitted by me to the Department of Software Engineering, Delhi Technological University, Delhi in partial fulfilment of the requirement for the award of degree of Master of Technology, is original and not copied from any source without proper citation. This work has not previously formed the basis for the award of any Degree, Diploma Associateship, Fellowship or other similar title or recognition.

Place: Delhi

Date: 30/05/2023

  
**Saransh Sharma**

2K21/SWE/22

## DECLARATION

We/I hereby certify that the work which is presented in the Major Project-II/Research Work entitled Automatic Comment Generation Using Deep Learning Techniques in fulfilment of the requirement for the award of the Degree of Bachelor/Master of Technology in Software Engineering and submitted to the Department of Software Engineering, Delhi Technological University, Delhi is an authentic record of my/our own, carried out during a period from 2021-2023, under the supervision of Ms. Priya Singh.

The matter presented in this report/thesis has not been submitted by us/me for the award of any other degree of this or any other Institute/University. The work has been published/accepted/communicated in SCI/ SCI expanded/SSCI/Scopus indexed journal OR peer reviewed Scopus indexed conference with the following details:

Title of the Paper: A literature Review on Automatic Comment Generation using Deep Learning Techniques  
Author names (in sequence as per research paper): Priya Singh and Saransh Sharma  
Name of Conference/Journal: 2022 3<sup>rd</sup> International Conference on Issues & Challenges in Intelligent Computing Techniques (ICICT)  
Conference Dates with venue (if applicable): 11 November 2022  
Have you registered for the conference (Yes/No)?: Yes  
Status of paper (Accepted/Published/Communicated): Published  
Date of paper communication: 30<sup>th</sup> September 2022  
Date of paper acceptance: 21<sup>st</sup> October 2022  
Date of paper publication: 20<sup>th</sup> March 2023

Saransh  
Saransh Sharma  
2K21/SWE/22  
Student(s) Roll No., Name and Signature

## SUPERVISOR CERTIFICATE

To the best of my knowledge, the above work has not been submitted in part or full for any Degree or Diploma to this University or elsewhere. I, further certify that the publication and indexing information given by the students is correct.

Place: Delhi

Date: 29/05/2023

  
Supervisor Name and Signature

Ms. PRIYA SINGH  
ASSISTANT PROFESSOR

**NOTE: PLEASE ENCLOSE RESEARCH PAPER ACCEPTANCE/ PUBLICATION/COMMUNICATION PROOF ALONG WITH SCOPUS INDEXING PROOF (Conference Website OR Science Direct in case of Journal Publication).**

**DEPARTMENT OF SOFTWARE ENGINEERING**

**DELHI TECHNOLOGICAL UNIVERSITY**

(Formerly Delhi College of Engineering)

Bawana Road, Delhi-110042

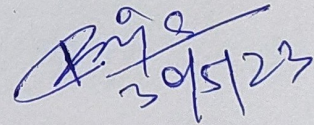
**CERTIFICATE**

I hereby certify that the Project Dissertation titled “**Automatic Comment Generation using Deep Learning Technique**” which is submitted by Saransh Sharma, Roll No – 2K21/SWE/21, Department of Software Engineering, Delhi Technological University, Delhi in partial fulfilment of the requirement for the award of the degree of Master of Technology, is a record of the project work carried out by the student under my supervision. To the best of my knowledge this work has not been submitted in part or full for any Degree or Diploma to this University or elsewhere.

Saransh Sharma  
2K21/SWE/21

Place: DTU, Delhi

Date: 30/05/2023



**Ms. PRIYA SINGH**

**SUPERVISOR**

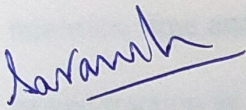
(Assistant Professor, SE, DTU)

## ACKNOWLEDGEMENT

I'm extremely grateful to Ms Priya Singh (Assistant Professor, Department of Software Engineering) and all the employees of the Department of Software Engineering at DTU. They all furnished us with enormous help and direction for the venture.

I might likewise want to offer my thanks to the University for giving us the research centres, framework, testing offices and climate which permitted us to work with next to no checks.

I might likewise want to see the value in the help gave to us by our lab partners, seniors and our companion bunch who supported us with all the information they had in regards to different subjects.



Saransh Sharma  
2K21/SWE/22

## **ABSTRACT**

Comment Generation is an emerging field that has attracted significant attention from researchers. They are actively exploring various methods to address this problem and enhance the effectiveness of comment generation. Deep learning techniques have gained popularity in recent times as researchers leverage them to tackle this challenge. The 21st century has witnessed a substantial increase in software development, resulting in a significant rise in lines of code. To expedite the process of writing comments for these codes, individuals are endeavouring to develop automated comment generation systems. Such systems aim to provide comprehensive comments within the source document, enabling new developers to quickly comprehend and commence their work. In the competitive industry, organizations strive to minimize time and maximize productivity. Consequently, comment generation has gradually gained traction as an intriguing area of interest for researchers.

# CONTENTS

<b>TABLE OF CONTENTS</b>	<b>PAGE NO</b>
<b>DECLARATION</b>	<b>i</b>
<b>CERTIFICATE</b>	<b>ii</b>
<b>ACKNOWLEDGEMENT</b>	<b>iii</b>
<b>ABSTRACT</b>	<b>iv</b>
<b>CONTENTS</b>	<b>v</b>
<b>LIST OF TABLES</b>	<b>vii</b>
<b>LIST OF FIGURES</b>	<b>viii</b>
<b>LIST OF ABBREVIATIONS</b>	<b>ix</b>
<b>CHAPTER 1 INTRODUCTION</b>	<b>1</b>
<b>1.1. BACKGROUND AND MOTIVATION</b>	<b>1</b>
<b>1.2. AUTOMATIC COMMENT GENERATION</b>	<b>3</b>
<b>1.3. RESEARCH OBJECTIVE</b>	<b>5</b>
<b>1.4. ORGANISATION OF THESIS</b>	<b>5</b>
<b>CHAPTER 2 LITERATURE SURVEY</b>	<b>6</b>
<b>2.1 RESEARCH QUESTIONS FORMATION</b>	<b>6</b>
<b>2.2 SEARCH STRATEGY AND STUDY SELECTION</b>	<b>7</b>
<b>2.3 DATA EXTRACTION AND SYNTHESIS</b>	<b>7</b>
<b>2.4 DESCRIPTION OF PRIMARY STUDIES AND QUALITY ANALYSIS</b>	<b>8</b>
<b>2.5 RELATED WORK</b>	<b>9</b>
<b>CHAPTER 3 RESEARCH METHODOLOGY</b>	<b>20</b>
<b>3.1 EXPERIMENTAL SETUP</b>	<b>20</b>
<b>3.1.1 HARDWARE USED</b>	<b>20</b>
<b>3.1.2 SOFTWARE USED</b>	<b>20</b>
<b>3.2 SAMPLE DATASET</b>	<b>20</b>
<b>3.3 TECHNIQUE USED</b>	<b>21</b>
<b>CHAPTER 4 RESULTS</b>	<b>27</b>
<b>CHAPTER 5 LIMITATIONS</b>	<b>29</b>

<b>CHAPTER 6 CONCLUSION AND FUTURE SCOPE</b>	<b>30</b>
<b>REFERENCES</b>	<b>32</b>
<b>APPENDICE</b>	<b>34</b>
<b>A. PLAGIARISM REPORT</b>	<b>34</b>
<b>B. LIST OF PUBLICATIONS</b>	<b>35</b>



## List of Tables

<b>Table Number</b>	<b>Figure Name</b>	<b>Page Number</b>
2.1	Research Questions	6
2.2	Quality Assessment Questions	8
2.3	Summarization recent models	12
2.4	Comparison of Models	13
2.5	Merits and De-Merits of DLTs	19

## List of Figures

<b>Figure Number</b>	<b>Figure Name</b>	<b>Page Number</b>
1.1	Deep Learning w.r.t ML & AI	1
1.2	Performance of DL & ML	2
1.3	Multilayer Network	2
1.4	Computing on Neural Network	3
1.5	Comment Generation Process	4
1.6	Comment Generation Framework	4
2.1	Code RNN	10
2.2	Workflow of model	11
2.3	Architecture of Deep Code	12
2.4	Basic Seq2Seq	14
2.5	Encoder	14
2.6	Decoder	15
2.7	LSTM Logic	15
2.8	CNN in code generation	16
2.9	RNN	16
2.10	DLT	17
3.1	Encoder Class	22
3.2	Decoder Class	23
3.3	Training Model	25
4.1	BLEU for comments	28
4.2	BLEU for codes	28

## LIST OF ABBREVIATIONS

<b>Abbreviation</b>	<b>Definition</b>
<b>CNN</b>	<b>CONVOLUTIONAL NEURAL NETWORK</b>
<b>RNN</b>	<b>RECURRENT NEURAL NETWORK</b>
<b>LSTM</b>	<b>LONG SHORT-TERM MEMORY</b>
<b>GRU</b>	<b>GRATED RECURRENT UNITS</b>
<b>BLEU</b>	<b>BILINGUAL EVALUATION UNDERSTUDY</b>
<b>ROUGE</b>	<b>RECALL-ORIENTED UNDERSTUDY FOR GISTING EVALUTION</b>

# CHAPTER 1

## INTRODUCTION

### 1.1 BACKGROUND AND MOTIVATION

Deep learning is a system designed to mimic the way the human brain learns. It draws inspiration from the architecture of the brain, where neurons are the fundamental building blocks. Similarly, deep learning architecture includes computational units that can model nonlinear functions. The key to deep learning lies in the neurons, which, like their biological counterparts, receive input signals and generate output signals. These neurons are organized into layers, with each layer responsible for identifying specific patterns in the data. By stacking multiple layers, the system learns to understand data representations. This layered structure resembles the network of neurons in the brain, leading to the term "neural networks" or "artificial neural networks" to describe the deep learning architecture.

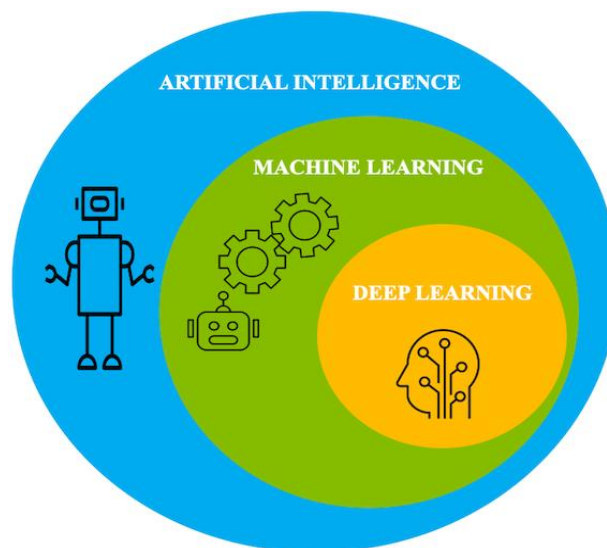


Figure 1.1 Deep Learning w.r.t ML and AI

As you can see in figure 1, [6] that deep learning is basically a subset of machine learning which itself is subset of Artificial Intelligence.

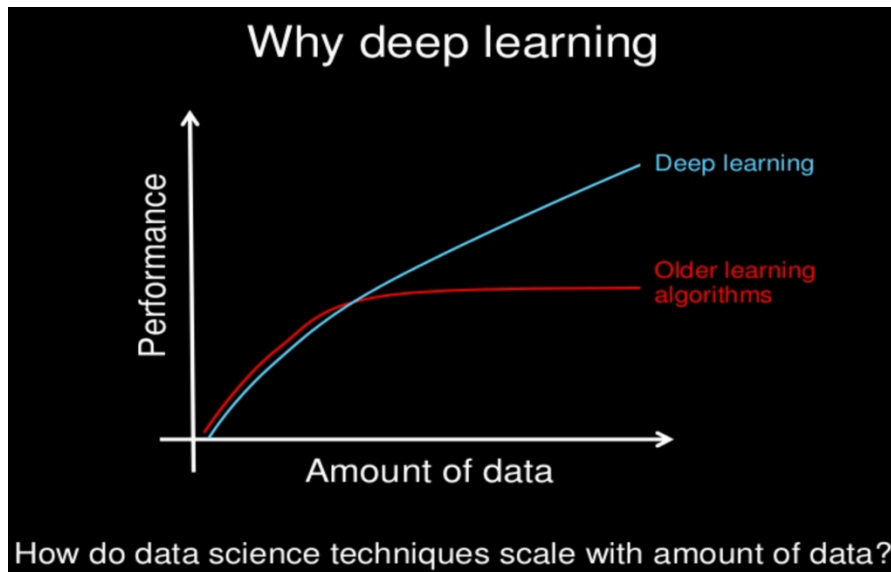


Figure 1.2 [5] Performance of DL and ML

As, we can see from above diagram that as the [5] size of data is increasing, the older machine learning algorithms performance is not increasing with the size of data sets. But with deep learning algorithms as the size of data sets increases, its ability to learn from it increases and hence its performance increases as the data size increases.

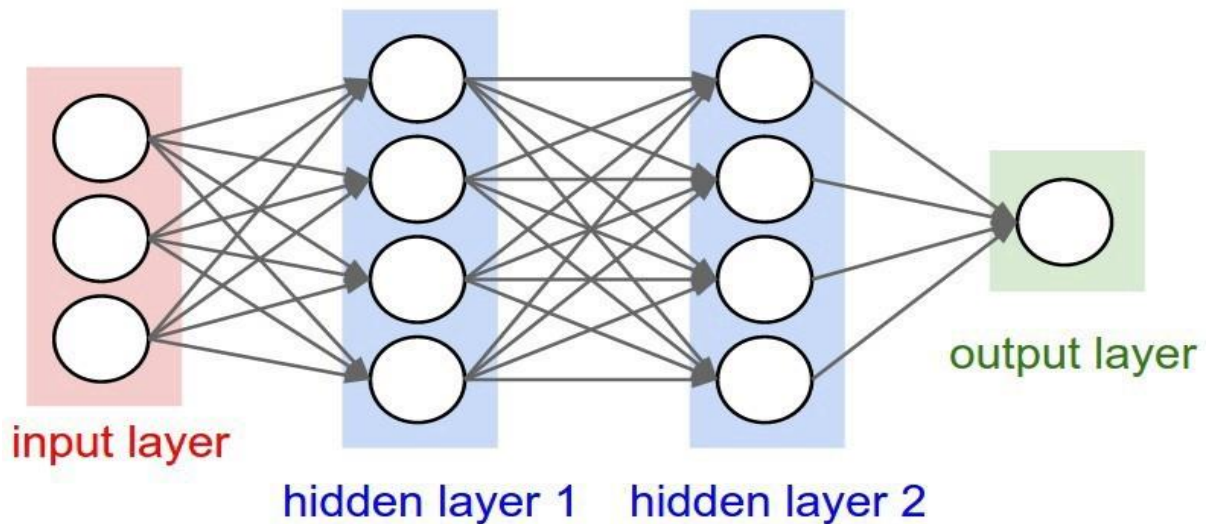


Figure 1.3 [7] Multilayer Network

As we see in figure 3, [7] basically input is your features and hidden layers are where the computation take place.

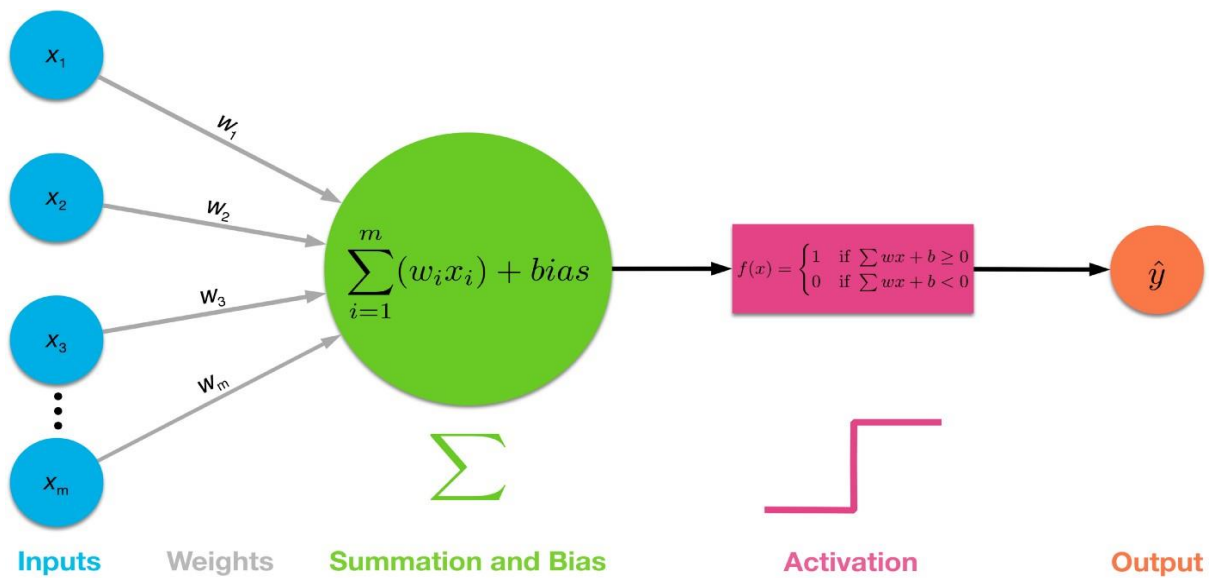


Figure 1.4 [4] Computing on Neural Network

For example, in image classification using logistic regression neural network, we need to find parameters  $w$  and  $b$  so that we can get better result. In figure 4, [4] we can see that our input is actually at 0-level, summation is at 1-level, activation is at 2-level. Also called 2-layer neural network. Different techniques can be utilized to make solid Deep learning models. These procedures incorporate learning rate rot, move picking up, preparing without any preparation and dropout. For this we have different algorithms of deep learning. Various scenarios are there and each procedure is suitable according to the situations.

## 1.2 AUTOMATIC COMMENT GENERATION

Comment generation is basically, generating comments in the source documents so that any user can read it and understands the document comfortably. Many techniques are used for generating the comments. Recently, people are started focusing on this problem as nowadays we have code which are of millions of lines. So, when you have such huge number of lines of codes, comment generation play an important role to understand the document and also it reduces the time of developer in writing the comments manually.

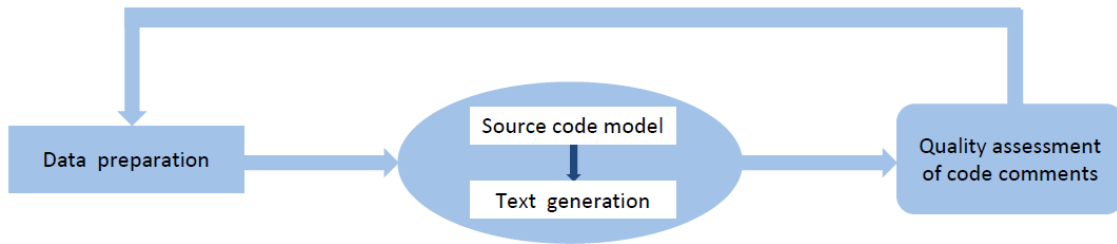


Figure 1.5 [26] Comment Generation Process

As we can see in the above diagram, the process for generating comments. It is very important to note that data needs to be prepared properly and should follow standard coding rules and procedures. If the data is prepared properly and scaled effectively then efficiency of the technique to generate the comment can be increased significantly. So, once the data is prepared properly then one choose technique accordingly. Till now researches have many techniques but since the evolution of deep learning, now researchers are shifting on deep learning techniques to generate the comments. Deep learning is used because its efficiency increases with the increase in data set. And as we all know how rapidly software development is increasing in today’s world. Hence, using deep learning is a good way to tackle the problem of comment generation.

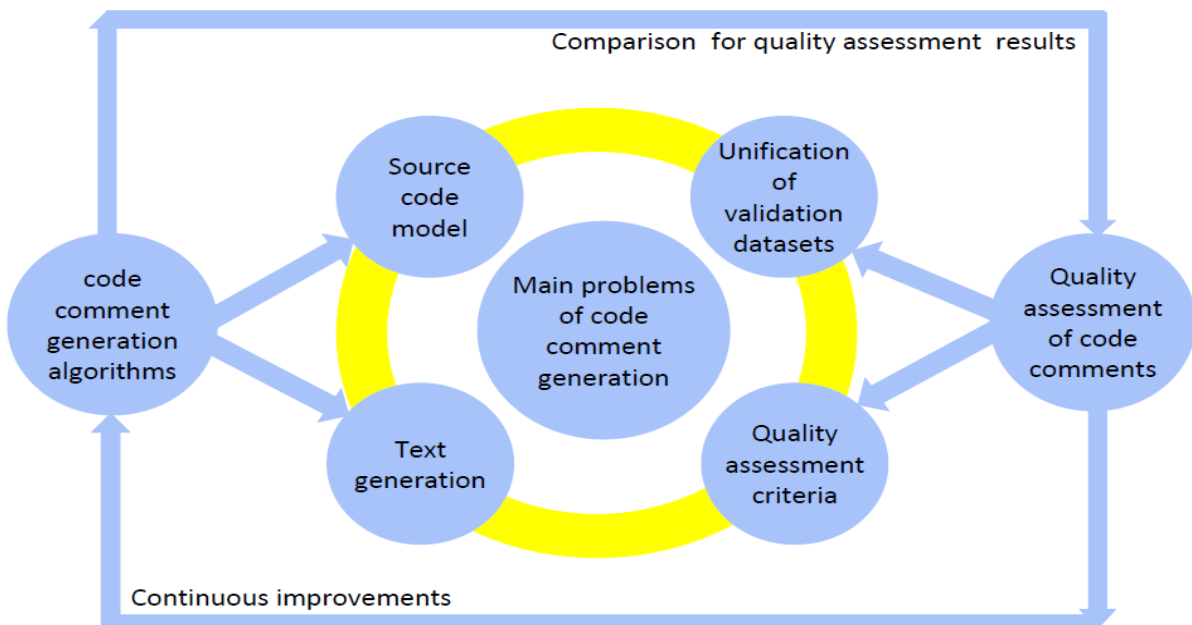


Figure 1.6 [26] Comment Generation Framework

By seeing above diagram, we can say that generating comments is not the only problem that we are facing, but generating comments with accuracy is a problem. If comments generated are not meaningful then it is of no use. Hence, we need to take care of the accuracy of the generated comments about how correctly they are defining the functions. There should be many questions that must arise in the mind about how to ensure the accuracy of the quality of comments that are generated. To cover this, we need a big data set comprising thousands of snippets of code so that our deep learning model can be trained properly. This is the advantage of using deep learning. Its algorithms learn better and give better performance if the data set is large.

### **1.3 RESEARCH OBJECTIVE**

Objective of the research is to use deep learning techniques and find a way to generate comments for a code snippet with a better accuracy. With this, we will be able to reduce the time for an organization and especially for the developer to not to waste time in writing comments in a code manually. Using deep learning techniques, we will try to achieve this and also takes care of the accuracy of the generated comment. Because if comments are not meaningful then it is of no use. So, the objective is to do comment generation. We will first try to do this on a particular language until we get good accuracy and then we can explore and do it for others also and try to create a generalise model which can also be a future work depending on the success we get.

### **1.4 ORGANIZATION OF THESIS**

The rest of the thesis is divided into the following Chapters.

**Chapter 2** consists of a discussion of Literature review of our own work in the field of automatic comment generation represents **Chapter 3** discusses the methodology for the review where we mention the research questions formed, and the search strategy we have adopted following that the exclusion and inclusion criteria. **Chapter 4** shows the results of our implementations **Chapter 5** talks about the limitations of automatic comment generation which are still a major drawback **Chapter 6** concludes the paper and mentions the future scope.



# CHAPTER 2

## LITERATURE SURVEY

This systematic literature review follows the guidelines outlined by Kitchenham [21], which outlines three distinct phases: (1) Planning, (2) Conducting the review, and (3) Reporting. Each phase holds significance in the process of writing a systematic literature review. During the planning phase, the research questions are defined. In the second phase, conducting the review, relevant research is identified, primary studies are selected, and their quality is assessed. Finally, in the third phase, the review is written based on the research conducted in the first and second phases.

### 2.1 Formulation of Research Questions

As advancements continue to occur in comment generation techniques, two aspects that remain relevant are data extraction and deduction, as well as comment generation utilizing deep learning methods. To gain a deeper understanding of these topics, the following research questions have been identified:

Table 2.1: Research Questions and Objectives

S. No	Research Question	Objective
1	Which DLT (deep learning techniques) are utilized for automatic comment generation in source-code (ACG-SC) in the source code?	To know the different DLT used in ACG-SC.
2	What performance measures are used to evaluate the performance of DLT applied for ACG-SC?	To know which DLT is able to give better results.
3	Which datasets have been used across DLTs for ACG-SC?	To identify various types of data sets used in ACG-SC.
4	What are the merits and demerits of using different DLTs for ACG-SC?	To understand when and where which DLT can be used.

## **2.2 SEARCH STRATEGY AND STUDY SELECTION**

The selection of primary studies is a crucial step in the second phase of the Kitchenham guidelines. It ensures that only high-quality papers meeting our criteria are included, while excluding those that do not align with our objectives. We conducted searches in the following digital libraries:

- ScienceDirect
- ACM Digital Library
- IEEE Xplore
- Springer Link

Inclusion:

1. Empirical research utilizing deep learning techniques.
2. Literature specifically focusing on comment generation using deep learning techniques.
3. Literature authored by students, researchers, and professional software developers.
4. Literature written in the English language.

Exclusion:

1. Non-empirical studies.
2. Literature not directly related to comment generation using deep learning techniques.
3. Literature authored by individuals outside the scope of students, researchers, and professional software developers.
4. Literature not written in the English language

## **2.3 DATA EXTRACTION AND SYNTHESIS**

To enhance the accuracy of our research question answers, we have extracted the following information from the primary studies:

- Methodology employed in the research
- Publication year
- Type of dataset used
- Pre-processing technique utilized
- Primary input utilized
- Technique employed for comment generation
- Criteria employed for evaluation

After doing data extraction and using quality assessment questions, we selected 20 papers out of 25 selected papers. Excluded 5 papers to not matching the standard set of questions we created to include only related papers of DLT.

## 2.4 DESCRIPTION OF PRIMARY STUDIES AND QUALITY ANALYSIS

For getting better results, it's important to have quality assessment questions. Hence, we designed 10 quality questions as shown in Table 1. for our primary studies to get preeminent quality of papers.

Table 2.2: Quality Assessment Questions

S.	Question	Yes	Partly	No
1	Is the objective stated prominently?			
2	Is the method of study stated clearly?			
3	Whether data selection, extraction and pre-processing of data described clearly?			
4	Do the study provide related literature?			
5	Are the Research Questions formed accordingly?			
6	Whether the study clearly justifies the importance of prediction technique used?			
7	If more than one technique is used then comparative study done or not?			
8	Are the constraints of study stated clearly?			
9	Are results defining properly?			
10	Does performance measures used for evaluating the predictive performance in the study?			

## 2.5 RELATED WORK

Comment Generation is a new field but if we see the history then people started working on it since 2010. Various methods are being used to generate the comments. But recently people have shifted on using deep learning methods. Earlier, IR based comment generation algorithms were supposed to being used which uses various algorithms like latent semantic indexing, vector space model and other algorithms. But problem was that they were not able to produce better accuracy. Comments were generated but it was not meaningful. In recent years especially since 2016, researched shifted on deep learning and started using deep learning algorithms. One such work was done by [4] Yuding who came up with recurrent neural network. Basically, recurrent neural network is used when problem is related to sequence. For example, saying how are you looks good but if your model predicts are how you then it's not right. So, to solve such kind of problems recurrent neural network are used. Some researchers came up with long short-term memory technique which you can say is a type of recurrent neural network only but it uses the concept of encoders and decoders. There should be many questions that must arise in the mind about how to ensure the accuracy of the quality of comments that are generated. To cover this, we need a big data set comprising thousands of snippets of code so that our deep learning model can be trained properly. This is the advantage of using deep learning. Its algorithms learn better and give better performance if the data set is large. Till now researches have many techniques but since the evolution of deep learning, now researchers are shifting on deep learning techniques to generate the comments. Deep learning is used because its efficiency increases with the increase in data set. And as we all know how rapidly software development is increasing in today's world. Hence, using deep learning is a good way to tackle the problem of comment generation. Recently, researchers are now moving with deep learning techniques since its able to generate better accuracy. Many models are being used like recurrent neural network model, seq2seq model, LSTM model. These models have given better accuracy then previous techniques. Hence, researches are taking more interest in doing comment generation with deep learning techniques.

A comparative study was done to understand the current problems and results of various techniques of comment generation so that we can come up with better approach and with more features. We will see some of the recent techniques now to understand it.

## Comment Generation via Code RNN:

It is very hard to read long codes if comments are not present in the source document. To solve this, [4] Yuding came with code RNN approach which can perform better than other moles to generate comments more accurately. He used recurrent neural network technique of deep learning to overcome this problem.

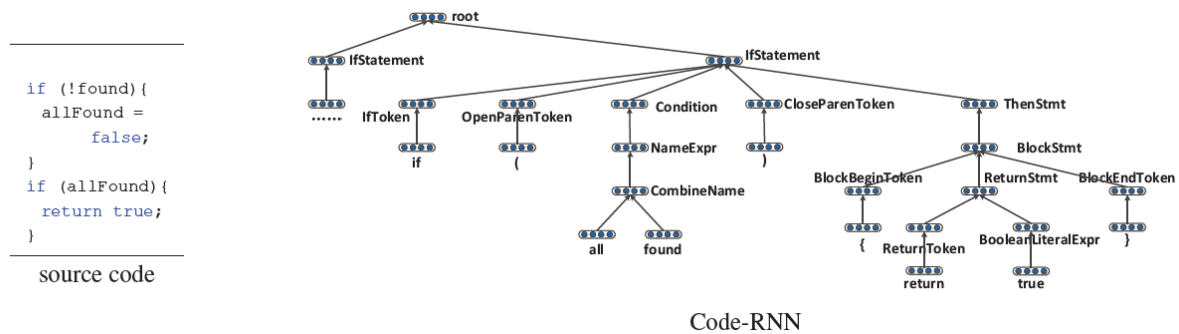


Figure 2.1[2] Code RNN

In this, every subtree of a program is parsed into neural network and represent it as a vector. Now they break long data into meaningful words. And used the technique of reversive neural network to train it. So, this approach of recurrent neural network is used because we know our output needs to be in correct sequence. So, whenever we know output has to be in sequence, in that case building a model with recurrent neural network can be a good choice.

## Comment Generation via Deep Reinforcement Learning:

Comment Generation using deep reinforcement learning is another way to generate the results. It basically uses the concept of [2] LSTM (long short-term memory) which basically is a type of recurrent neural network and it over comes one problem of recurrent neural network that is recurrent neural network suffers from the long input problem so long short-term memory overcomes this. This work can be done with better accuracy in long short-term memory then recurrent neural network.

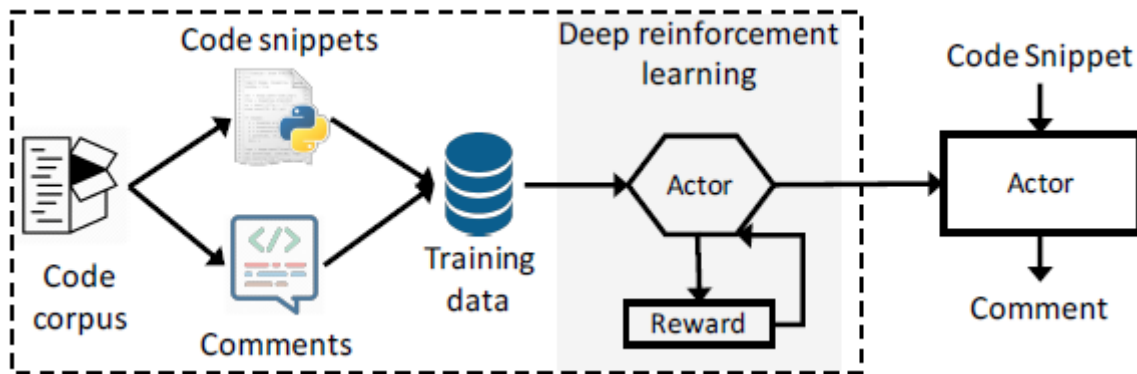


Figure 2.2[2] Workflow of model

Basically, by using deep reinforcement learning what happens is that the system learns with the experience and by combining it with deep learning, it behaves totally like a brain behaves, it keeps on learning. But as soon as we connect both the concept, the problem arises as deep learning basic requirement is to have large amount of data so when we use deep reinforcement then we need large amount of data. If we don't have the large amount of data then it will give less accuracy as model won't be trained properly.

Comment Generation via A Deep Code Comment Generation Tool with Hybrid Lexical and Syntactical Information:

[1] Model consists of three stages:

- i) Data processing: The Java techniques acquired from GitHub are parsed into equal corpus. The objective remarks are removed from the relating Javadoc of the Java techniques. To get familiar with the primary data, the Java techniques are changed over to AST arrangements before took care of into the model.
- ii) Model training: We utilize Tensorflow, which is an open-source profound learning system, to construct our models.
- iii) Online testing: Both Information Retrieval (IR) metrics (e.g., precision, recall, F-score and F-mean) and Machine Translation (MT) metrics (e.g., BLEU and METEOR) are used to evaluate the model.

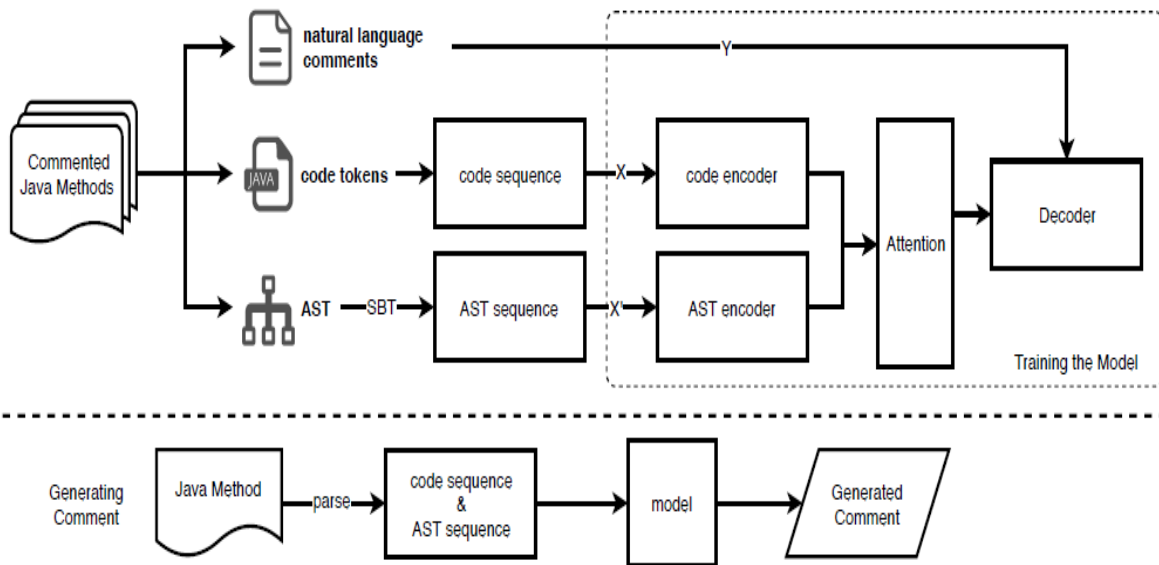


Figure 2.3[4] Architecture of Deep Code

As we see from the above architecture of this model, the input that is sample java methods are parse from which code sequence and AST sequence are being generated which is being fetched into their model and then comments are generated. Generated comments are then compared with the actual comments to see the accuracy of the model. Here both encoder decoder is used for the training the model as visible from the architecture easily. Attention is also used to focus on a particular thing to try to improve the accuracy of the generated comments. To summarise, we have pointed out the advantages and disadvantages of each approach. Each approach is applicable according to what the requirements of the user is.

Table 2.3 Summarization of recent models

Author	Yuding Liang	Yao Wan	Boao Li
Paper	Comment Generation via Code RNN	Comment Generation via Deep Reinforcement Learning	Comment Generation via A Deep Code Comment Generation Tool with Hybrid Lexical and Syntactical Information
Technique	Recurrent Neural Network	Reinforcement learning with deep learning	Deep neural network combines lexical syntactical information
Advantages	Good for generating comments just for	Solves two problems that is code	Uses attention mechanism which

	functions in the code script, take long inputs easily	representation and exposure bias	learns lexical and structural information, gives better accuracy.
<b>Disadvantages</b>	RNN requires large amount of data set, processing can be slow because its recurrent.	Uses LSTM which are prone to overfitting, states overloading can happen.	Similar to seq2seq model where if long sequences are there then it can lose the initial context, pretrained data needs to be large.

As we can see from the above table, each model has their own advantages and disadvantages. So, according to user requirements one can select what model will work for them. But still a generalise model is currently missing in the industry which can really help all the developers so that all the manual work of writing comments can be done automatically.

## Results

After going through the above models, we found out that each model has their own advantages and disadvantages. So, we found out that rather than using recurrent neural network, one can use long short-term memory technique as it works better when long input is present. Below is the table which compares the models on ROGUE-2 values. Code-RNN models works better then Code-NN model and also from models which have used seq2seq models. This shows that Code-RNN is better when compared with simple neural network.

As we can see from the above table that they used rogue-2 values to measure the efficiency of their technique so that they can compare it with other models. We can clearly see that their model is performing better if rogue-2 criteria is taken to measure the performance. Here, we can see that Deep Code is working better then seq2seq by using hybrid2Seq with Attention and DRL methods.

Table 2.4 comparison between different models

	BLEU-1	BLEU-2	BLEU-3	BLEU-4	METEOR	ROUGE-L	CIDER
Seq2Seq+Attn+DRL	0.2421	0.0919	0.0513	0.0325	0.0882	0.3935	0.6390
Tree2Seq+Attn+DRL	0.2309	0.0854	0.0499	0.0338	0.0843	0.3767	0.6060
Hybrid2Seq	0.1837	0.0379	0.0183	0.0122	0.0604	0.3020	0.2223
Hybrid2Seq+Attn	0.1965	0.0516	0.0280	0.0189	0.0693	0.3154	0.3475
Hybrid2Seq+Attn+DRL (Our)	0.2527	0.1033	0.0640	0.0441	0.0929	0.3913	0.7501

As seen, using deep reinforcement learning, they were able to achieve better performance compared to other models in their domain. They used BLEU score as their criteria to evaluate.



RQ1: Which DLT (deep learning techniques) are utilized for automatic comment generation in source-code (ACG-SC) in the source code?

The objective of this RQ is to find the different techniques that are being used till now in ACG-SC. We were successfully able to find out the different DLTs that are being used from 2016-2021. After analysing the 20 research papers which used deep learning techniques from 2016-2021, we found out the following results:

*Seq2Seq*: This DLT is used for 35% of times, it's basically a type of RNN but with an encoder-decoder architecture where both are LSTM [23].

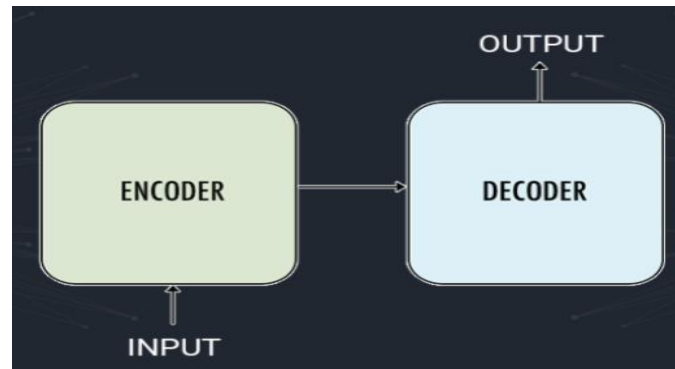


Figure 2.4 [24]: Basic Seq2Seq

In this process, the encoder takes an input and combines it into hidden state and cell state vectors. The hidden state is calculated, while the output of the encoder is disregarded, and the focus is solely on the internal state of the encoder.

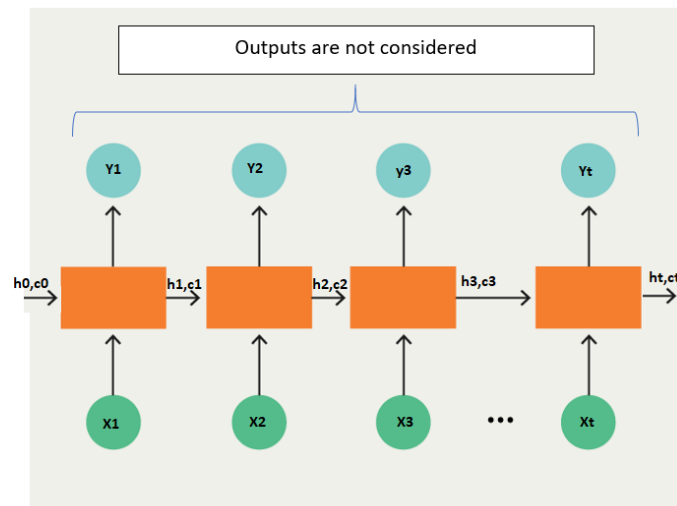


Figure 2.5 [24]: Encoder

In this process, we observe a sequential passing of each input to the subsequent time step, with no consideration given to the output of each step. As previously mentioned, only the internal state denoted by 'h' and 'c' in Figure 2 holds significance. Now, let's examine the role of the decoder. In this context, the final state of the encoder is transferred to the initial state of the decoder. Consequently, the initial states of the decoder are established based on the concluding

states of the encoder. Subsequently, the decoder commences the output phase, and these outputs are also utilized for future outputs.

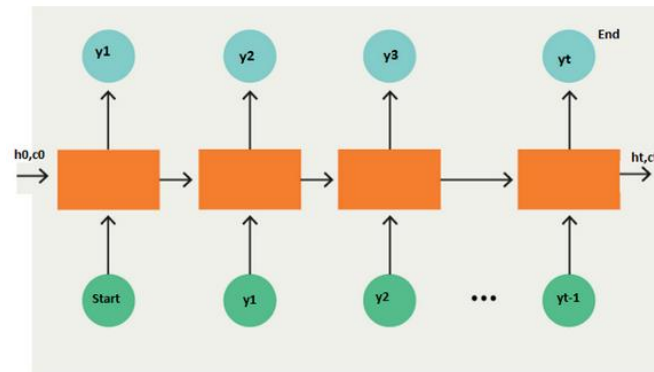


Figure 2.6 [24]: Decoder

The seq2seq model functions at a fundamental level and has gained significant popularity for generating comments in source code. It is currently preferred over alternative methods. Approximately 25% of the time, the model employs LSTM (Long Short-Term Memory) as a technique. LSTM incorporates a memory unit that enables it to store and preserve information instead of merely transferring the output to the next part of the network.

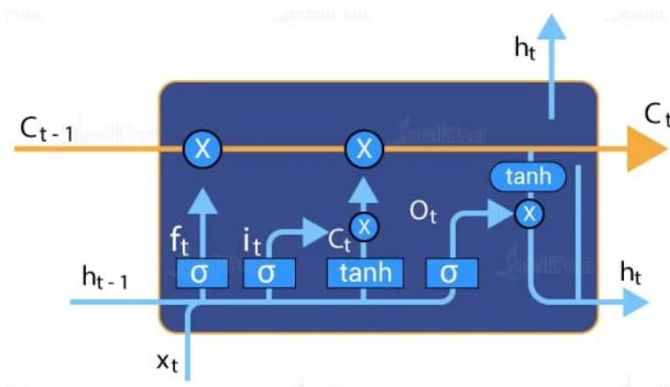


Figure 2.7 [22]: LSTM logic

LSTM (Long Short-Term Memory) networks possess the ability to learn long sequences due to their memory cell, also known as the cell state. This cell state ensures the maintenance of information over time, resembling a conveyor belt through which data flows. When dealing with lengthy comments or sequences, LSTM can be a preferable choice due to its capacity to handle them effectively.

CNN (Convolutional Neural Network) is another deep learning technique used approximately 15% of the time. One advantage of CNN is that it requires relatively less pre-processed data compared to other techniques.

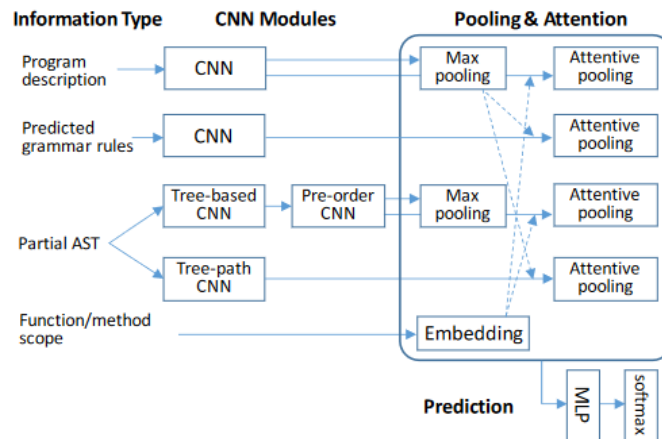


Figure 2.8 [25]: CNN in code generation

In his research paper, Zeyuusun [25] employed CNN (Convolutional Neural Network) to generate code based on a well-defined structure. While pooling is a step commonly used in implementing CNN, attentive pooling allows the network to focus on specific patterns, enabling code generation if the structure is well-defined. However, a drawback arises when the input is not provided according to the expected structure, leading to incorrect results and rendering the model ineffective. In such cases, it suffers due to the input not aligning with the intended structure.

RNN (Recurrent Neural Network) is another deep learning technique used approximately 15% of the time. This approach involves saving the output and feeding it back as input to the next part of the network.

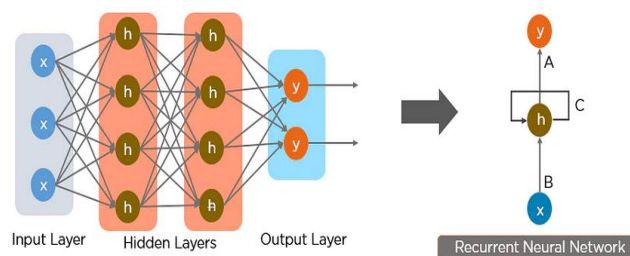


Figure 2.9 [15]: RNN

With the advancement of recurrent neural networks (RNN), other neural networks such as LSTM and Seq2Seq have emerged. Both LSTM and Seq2Seq are types of RNN that build upon the fundamental concept of RNN. The core idea behind RNN is to store the output of a layer and feed it back as input to obtain the layer's output.

GNN (Graph Neural Network) is a type of deep learning technique used approximately 5% of the time. It is employed when the objective is to predict outputs without prior knowledge of the ground truth.

Transformer, another type of deep learning technique, is also used around 5% of the time. It incorporates self-attention mechanisms to selectively assign importance to different parts of the input.

Overall, LSTM, Seq2Seq, GNN, and Transformer are all variations of neural networks that have their own specific applications and utilization frequencies.

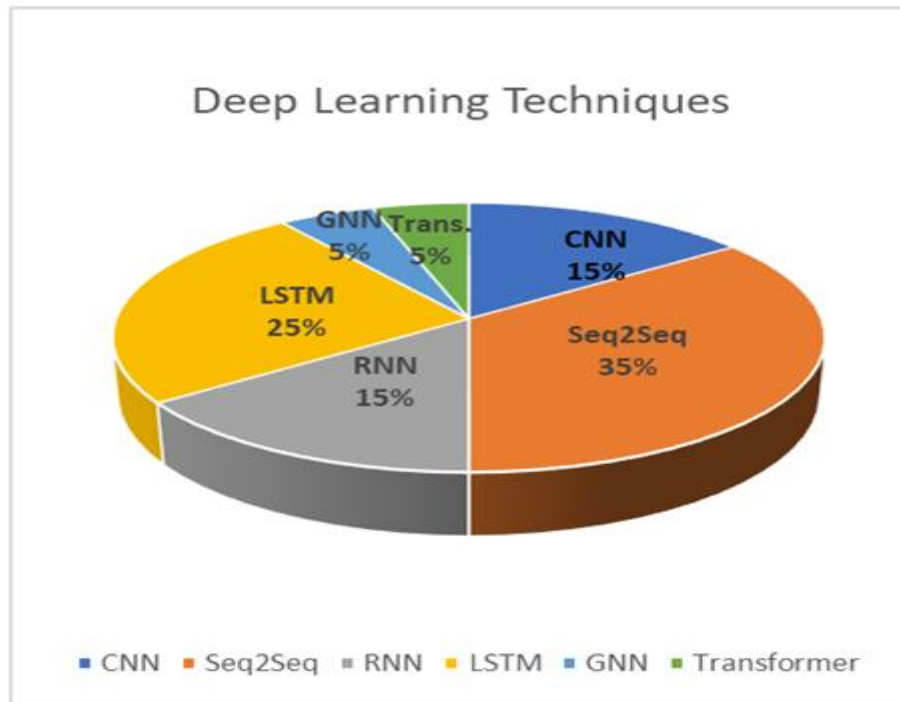


Figure 2.10: DLT

RQ2: What performance measures are used to evaluate the performance of DLT applied for ACG-SC?

Automated machine translation evaluation metrics are utilized to assess the effectiveness of applying DLT for ACG-SC. These metrics include:

1. BLEU Score: This metric quantifies the similarity between the system output and the reference translations by counting the occurrence of n-grams or word sequences.
2. METEOR Score: METEOR addresses the limitations of BLEU by considering both precision and recall scores, merging them through several stages of matching words that were not previously matched.
3. ROUGE Score: ROUGE is employed to evaluate automatically generated comments by comparing them with a set of reference translations.

Automated machine translation evaluation metric are frequently used in ACG-SC because they provide a more comprehensive understanding of performance compared to other techniques.

RQ3: Which datasets have been used across DLTs for ACG-SC?

The majority of datasets used in research are either sourced from open repositories or obtained from GitHub. However, a common challenge is that these datasets are often tailored to specific problems, making it difficult to compare and determine which model is superior to others. Without a standardized dataset, it is not currently possible to make conclusive judgments about the superiority of different techniques. In order to assess the performance of ACG-SC (presumably a specific technique), it is necessary to use a standardized dataset and conduct empirical research based on the methods mentioned in RQ1 (presumably a research question). Future work will involve selecting a standard dataset and conducting comprehensive research to evaluate the techniques.

RQ4: What are the merits and demerits of using different DLTs for ACG-SC?

The objective of this research question was to identify the advantages and disadvantages of different Distributed Ledger Technologies (DLTs). By doing so, the intention was to create a reference point for future researchers, enabling them to address existing issues and enhance the models effectively. The aim is to establish a foundation of knowledge that can inform and guide future research, facilitating improvements in DLTs based on the identified merits and demerits. Deep learning, particularly in the form of recurrent neural networks (RNNs) and generative models like GPT, has been utilized for automatic comment generation in various applications. While there are advantages to using deep learning for this task, there are also several disadvantages to consider. Deep learning models can generate comments that are often fluent and grammatically correct. They can learn to mimic the patterns and structure of human-generated comments, making the generated comments more coherent and natural-sounding. Deep learning models can capture and leverage contextual information from the input text to generate relevant comments. They can learn to infer meaning and respond appropriately based on the given context, leading to more contextually relevant comments. Deep learning models can benefit from large-scale training datasets. With abundant data, they can learn diverse language patterns and generate comments that cover a wide range of topics and styles. Deep learning models can automate the process of comment generation, saving time and effort compared to manual comment writing. They can generate comments at scale and provide immediate responses to user queries or prompts. Deep learning models often lack true creativity and originality. While they can generate coherent and contextually relevant comments, they often rely on patterns and phrases learned from the training data and may produce comments that are predictable or lack novelty. Deep learning models typically struggle with generating comments for inputs that are rare, out-of-domain, or significantly different from the training

data. They might produce irrelevant or nonsensical comments when faced with such inputs. Deep learning models learn from data that may contain biases, stereotypes, or offensive language. If not carefully designed and trained, the models can inadvertently generate comments that are inappropriate, offensive, or biased, potentially leading to negative consequences or harmful effects.

Table 2.5: Merits and De-Merits of DLTs

Technique	Merits	De-merits
Seq2Seq	Avoids the problem of vanishing gradient. It uses RNN or LSTM or GRU.	Pre-trained data needs to be large. If long sequences are there then it can lose initial context.
LSTM	Provides many inputs such as input/output biases, learning rates. No need for fine updates.	It is prone to overfitting. States overloading can happen.
RNN	It can take long inputs easily. Good for task which are time dependent.	Processing can be slow as it is recurrent. It requires large amount of data set.
CNN	It has less dependency on pre-processing. Easy and quick to implement.	Can be slow because of maxpool operation. It also requires large data set.
GNN	Easily able to capture graph data structure. Easy to train even with smaller data set.	Basic GNN are not able to correctly identify graph structure. Noise vulnerability in GNN exist.
Transformer	It helps in faster processing. All elements get equal attention.	Hard to control the attention in transformer. It can repeat or skip words.

# CHAPTER 3

## RESEARCH METHODOLOGY

### 3.1 EXPERIMENTAL SETUP

In this section, we will provide a concise overview of the hardware and software tools employed in our research. Additionally, we will delve into the performance measures, framework, and methodology utilized throughout the study.

#### 3.1.1 HARDWARE USED

The proposed project is based on ML approaches, which revolve around classification; the only hardware instrument required for implementation is a computer system. The model is created and run on a laptop with the given minimum hardware requirements.

- System Type Windows 10, Macintosh
- Processor Core i3 processor
- RAM 4GB
- Hard disk 500GB

#### 3.1.2 SOFTWARE USED

The Jupyter Notebook and Anaconda Navigator tools were used to implement each ML model. Python is a general-purpose, interpreted object-oriented programming language. It is a language that is open-source and free grown in popularity as a result of its condensed, straightforward, and extensive library support.

### 3.2 SAMPLE DATASET

Generating comments for Java code is a challenging task due to the structural and semantic complexities of the language. We explored the following options to gather a Java code dataset for comment generation:

1. **GitHub:** You can crawl Java repositories on GitHub and extract Java code along with associated comments. This process may require significant effort, but it can provide you with a diverse dataset for comment generation.
2. **JavaDoc:** Java code often includes comments in the form of JavaDoc, which provides documentation for classes, methods, and fields. You can collect JavaDoc comments from popular Java libraries or APIs to create a specialized dataset for comment generation.
3. **Stack Overflow:** Stack Overflow is a popular platform where developers ask and answer questions related to programming. You can search for Java code snippets on StackOverflow and extract the associated comments or discussions as a potential source for generating comments.
4. **Research Datasets:** Some academic research projects may have released datasets that include Java code and associated comments. You can explore research publications and repositories in the field of natural language processing (NLP) and code generation to find such datasets.

### **3.3 Technique Used**

We used torch library to implement our model. It is open-source library used for neural networks. In the above code, device is set to GPU if graphical processing unit is available else CPU will be used for tensor operations. This is the advantage of using PyTorch as its operations can be computed via CPU as well as from GPU. Optim module is imported from the torch.optim, it helps in gaining access to various optimization algorithms like stochastic gradient descent, Adam, RMSprop and more.



```

class Encoder(nn.Module):
    def __init__(self, input_dim, hidden_dim, embbed_dim, num_layers):
        super(Encoder, self).__init__()

        #set the encoder input dimesion , embbed dimesion, hidden dimesion, and number of layers
        self.input_dim = input_dim
        self.embbed_dim = embbed_dim
        self.hidden_dim = hidden_dim
        self.num_layers = num_layers

        #initialize the embedding layer with input and embbed dimation
        self.embedding = nn.Embedding(input_dim, self.embbed_dim)
        #intialize the GRU to take the input dimetion of embbed, and output dimation of hidden and
        #set the number of gru layers
        self.gru = nn.GRU(self.embbed_dim, self.hidden_dim, num_layers=self.num_layers)

    def forward(self, src):

        embedded = self.embedding(src).view(1,1,-1)
        outputs, hidden = self.gru(embedded)
        return outputs, hidden

```

Fig 3.1: Encoder Class

This code defines the Encoder class, which is a module in PyTorch used for encoding input sequences. The Encoder class is defined as a subclass of nn. Module, which is the base class for all neural network modules in PyTorch.

- In the `__init__` method, the constructor of the class, the input parameters are `input_dim`, `hidden_dim`, `embbed_dim`, and `num_layers`. These parameters represent the input dimension, hidden dimension, embedding dimension, and number of GRU layers for the encoder, respectively.
- The `super(Encoder, self).__init__()` line ensures that the initialization of the Encoder class inherits the properties and methods of the `nn.Module` class.
- The instance variables (`self.input_dim`, `self.embbed_dim`, `self.hidden_dim`, `self.num_layers`) store the values of the input dimensions, embedding dimension, hidden dimension, and number of layers for the encoder.
- The `self.embedding` variable initializes an embedding layer using `nn.Embedding`, which is used to convert input indices into dense vectors of fixed size (`embbed_dim`).
- The `self.gru` variable initializes a GRU (Gated Recurrent Unit) layer using `nn.GRU`. It takes the `embbed_dim` as the input size, `hidden_dim` as the hidden size, and `num_layers` to specify the number of GRU layers.
- The `forward` method defines the forward pass of the encoder. It takes an input tensor `src`, which represents the input sequence.

- The input tensor `src` is first passed through the embedding layer (`self.embedding(src)`) to convert it into dense embeddings. The `view(1, 1, -1)` reshapes the tensor to have dimensions (1, 1, -1) where the first 1 indicates the batch size (here, processing one sequence at a time) and -1 infers the remaining dimension based on the size of the input tensor.
- The reshaped tensor is then passed through the GRU layer (`self.gru(embedded)`). The GRU layer processes the input sequence and returns two outputs: `outputs`, which contains the output features from all time steps, and `hidden`, which represents the final hidden state of the GRU.
- Finally, the method returns `outputs` and `hidden` as the encoder's output.

Overall, this code sets up the Encoder class with an embedding layer and a GRU layer. It defines the forward pass to process input sequences and produce corresponding output features and hidden states. The specific dimensions and parameters of the encoder are specified during initialization.

```
class Decoder(nn.Module):
    def __init__(self, output_dim, hidden_dim, embed_dim, num_layers):
        super(Decoder, self).__init__()

        #set the encoder output dimension, embed dimension, hidden dimension, and number of layers
        self.embed_dim = embed_dim
        self.hidden_dim = hidden_dim
        self.output_dim = output_dim
        self.num_layers = num_layers

        # initialize every layer with the appropriate dimension. For the decoder layer, it will consist of an embedding, GRU, a Linear
        self.embedding = nn.Embedding(output_dim, self.embed_dim)
        self.gru = nn.GRU(self.embed_dim, self.hidden_dim, num_layers=self.num_layers)
        self.out = nn.Linear(self.hidden_dim, output_dim)
        self.softmax = nn.LogSoftmax(dim=1)

    def forward(self, input, hidden):

        # reshape the input to (1, batch_size)
        input = input.view(1, -1)
        embedded = F.relu(self.embedding(input))
        output, hidden = self.gru(embedded, hidden)
        prediction = self.softmax(self.out(output[0]))

        return prediction, hidden
```

Fig 3.2 Decoder Class

This code defines the Decoder class, which is another module in PyTorch used for decoding the hidden representation produced by the Encoder class.

- The Decoder class is defined as a subclass of `nn.Module`, similar to the Encoder class.
- In the `__init__` method, the constructor of the class, the input parameters are `output_dim`, `hidden_dim`, `embed_dim`, and `num_layers`. These parameters represent the output dimension, hidden dimension, embedding dimension, and number of GRU layers for the decoder, respectively.

- The instance variables (`self.embedded_dim`, `self.hidden_dim`, `self.output_dim`, `self.num_layers`) store the values of the embedding dimension, hidden dimension, output dimension, and number of layers for the decoder.
- The `self.embedding` variable initializes an embedding layer using `nn.Embedding`, which is used to convert output indices into dense vectors of fixed size (`embbed_dim`).
- The `self.gru` variable initializes a GRU layer using `nn.GRU`. It takes the `embbed_dim` as the input size, `hidden_dim` as the hidden size, and `num_layers` to specify the number of GRU layers.
- The `self.out` variable initializes a linear layer (`nn.Linear`) that maps the hidden dimension to the output dimension.
- The `self.softmax` variable initializes a log softmax activation function (`nn.LogSoftmax`) which applies logarithm and softmax operations to normalize the output probabilities along the specified dimension (`dim=1`).
- The `forward` method defines the forward pass of the decoder. It takes an `inputtensor` input and the previous hidden state `hidden`.
- The input tensor `input` is reshaped using `input.view(1, -1)` to have dimensions (1, `batch_size`). This allows processing a single time step for a batch of sequences simultaneously.
- The reshaped tensor is passed through the embedding layer (`self.embedding(input)`) and then through the ReLU activation function (`F.relu`) to obtain the embedded representation.
- The embedded tensor is then passed through the GRU layer (`self.gru(embedded, hidden)`), along with the previous hidden state `hidden`, to generate the output features and the updated hidden state.
- The output tensor is passed through the linear layer (`self.out(output[0])`) to map it to the output dimension.
- The resulting tensor is passed through the log softmax activation function (`self.softmax`) to obtain the prediction probabilities.
- Finally, the method returns the prediction probabilities and the updated hidden state.

Overall, this code sets up the Decoder class with an embedding layer, aaGRU layer, a linear layer, and a logsoftmax activationfunction. It defines the forward pass to process input tensors, generate output probabilities, and update the hidden state. The specific dimensions and parameters of the decoder are specified during initialization.

```

def trainModel(model, source, target, pairs, num_iteration=10):
    model.train()

    optimizer = optim.Rprop(model.parameters(), lr=0.01)
    criterion = nn.NLLLoss()
    total_loss_iterations = 0

    training_pairs = [tensorsFromPair(source, target, random.choice(pairs))
                      for i in range(num_iteration)]

    for iter in range(1, num_iteration+1):
        training_pair = training_pairs[iter - 1]
        input_tensor = training_pair[0]
        target_tensor = training_pair[1]

        loss = clacModel(model, input_tensor, target_tensor, optimizer, criterion)

        total_loss_iterations += loss

        if iter % 2 == 0:
            avarage_loss= total_loss_iterations / 100
            total_loss_iterations = 0
            print('%d %.4f' % (iter, avarage_loss))

    torch.save(model.state_dict(), 'generatemodel.pt')
    return model

```

Fig 3.3: Training Model

This code defines a function `trainModel` that trains a given model using a set of source-target pairs.

- The function `trainModel` takes the model to be trained, source and target as the language objects, pairs as a list of training pairs, and `num_iteration` to specify the number of training iterations.
- The model is set to training mode using `model.train()`.
- An optimizer (`optim.Rprop`) is instantiated with the model parameters and a learning rate of 0.01.
- A criterion (`nn.NLLLoss`) is instantiated for calculating the loss during training.
- A variable `total_loss_iterations` is initialized to keep track of the accumulated loss during iterations.
- `training_pairs` is a list comprehension that creates pairs of input and target tensors by calling `tensorsFromPair` function with random choices from pairs. This creates `num_iteration` training pairs.
- A loop iterates over the range from 1 to `num_iteration+1`.
- In each iteration, a training pair is retrieved from `training_pairs` based on the current iteration.

- The input tensor and target tensor are assigned from the training pair.
- The `clacModel` function is called with the model, input tensor, target tensor, optimizer, and criterion to calculate the loss.
- The loss is added to `total_loss_iterations` for tracking the cumulative loss.
- If the current iteration is divisible by 2, an average loss is calculated by dividing `total_loss_iterations` by 100 (assuming 100 iterations per average). The `total_loss_iterations` is then reset to 0.
- The iteration number and average loss are printed.
- After training iterations are completed, the trained model's state dictionary is saved to a file named 'generatemodel.pt' using `torch.save`.
- Finally, the trained model is returned.

# CHAPTER 4

## RESULTS

Comment generation for code is a challenging task due to the complex relationship between code and natural language. Generating meaningful and accurate comments that describe the functionality, purpose, and design choices of code requires an understanding of both the code structure and the desired comment style. Sequence-to-Sequence Models: These models treat comment generation as a language generation task, where the input is the code and the output is the corresponding comment. Sequence-to-sequence models, such as recurrent neural networks (RNNs).

It's worth noting that comment generation in Java, or any other programming language, is an active research area, and the performance of models heavily depends on the quality and diversity of the dataset used for training. Evaluating the performance of comment generation models requires metrics such as comment relevance, informativeness, and readability, which are subjective and can vary depending on the specific task and application.

BLEU score is typically calculated by comparing the n-grams (contiguous sequences of words) in the generated comment with those in the reference comment(s). It measures the overlap of these n-grams and assigns a score between 0 and 1, where a higher score indicates better similarity. However, BLEU does not capture semantic similarity or assess the quality of the generated comments in terms of informativeness or relevance.

For code comment generation, there might not be a direct one-to-one correspondence between generated comments and reference comments. The same code snippet can have multiple valid and diverse comments, making it challenging to define a single reference comment. Additionally, code comment generation often involves understanding the code's intent, functionality, and design choices, which may not be adequately captured by the BLEU metric. While BLEU can still provide some indication of comment generation quality, it is advisable to use additional evaluation metrics that focus on the quality and relevance of the generated

comments, such as human evaluation or task-specific metrics tailored to the application domain.

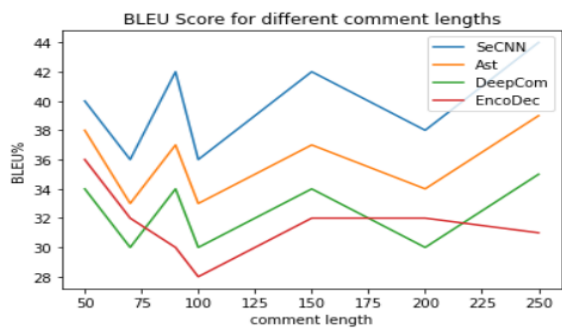


Fig 4.1 BLEU for comments

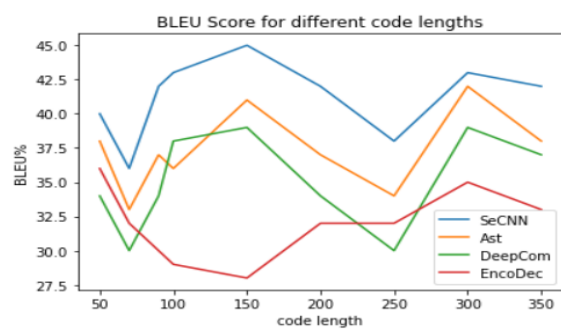


Fig 4.2 BLEU for code

# CHAPTER 5

## LIMITATIONS

After conducting a comparative study on comment generation, we discovered that each model showed improvements, but several assumptions were made during their development:

- Certain approaches assumed that comments would be generated specifically for functions and not for the entire code.
- Some methods successfully generated comments for variable declarations but struggled with generating comments for the entire code.
- Several models were capable of generating comments for the entire script, but their accuracy was notably low.
- Language limitations posed challenges, as certain approaches performed well for Java but not for other programming languages.
- The categorization of comments requires a systematic step-by-step process.

Considering these limitations, it becomes evident that there is still ample room for improvement in comment generation. Exploring alternatives such as modifying activation functions or combining deep learning with regular expressions could be considered. However, the existing limitations in current models raise concerns, particularly as scripts continue to grow in length. Thus, there is a need to develop a fool proof method for generating comments that can cover all aspects of a script and work effectively across multiple programming languages. As of now, a generalized model is still missing in this area of research.



# CHAPTER 6

## CONCLUSION AND FUTURE SCOPE

Comment generation has emerged as an area of great interest, with numerous researchers striving to develop improved approaches. As the IT industry undergoes digital transformation and advancements, the need for comprehensive code documentation becomes increasingly crucial. A model that provides high accuracy in generating comments can greatly facilitate code comprehension, enabling new developers to quickly understand and work on codebases by relying on the comments. Moreover, automating the process of writing comments can significantly reduce developer time and effort. In conclusion, the Seq2Seq model has demonstrated superior performance in automatically generating comments within source code. LSTMs are a type of recurrent neural network (RNN) specifically designed to handle long input sequences, which contributes to the Seq2Seq model's effectiveness in generating comments, particularly when dealing with lengthy comment text. Thus far, Seq2Seq models have shown prominence in the field of deep learning for comment generation.

To draw more definitive conclusions and gain deeper insights into the effectiveness of different techniques, further exploration is required, particularly for generating class-level comments and incorporating inline and outline comments. By expanding the scope of comment generation beyond method-level comments, a more comprehensive comparison of techniques can be conducted. With additional data and results, we can determine the state-of-the-art model for comment generation.

To enhance code readability, developers can incorporate both inline and outline comments. By employing regular expressions, one can generate comments for functions within the code, assuming the coder follows standard naming conventions. Existing research in this field has limitations, with some focusing solely on functions or variable declarations. It would be valuable to overcome these limitations and develop a comprehensive approach. Experimenting with different activation functions can potentially enhance accuracy, improving overall efficiency. Presently, there is no generalized model capable of generating comments for all

programming languages, indicating a need for further research. Furthermore, none of the existing models generate comments for every aspect of a code snippet.

We have categorized comment generation as follows:

1. **Comment Generation at the Method Level:** While most research has focused on this level, it is recommended to employ Deep Learning Techniques (DLT) in empirical studies, utilizing specific datasets to gain insights into which DLT methods perform better than others.
2. **Comment Generation at the Class Level:** Implementing DLT in source code for generating comments at the class level remains unexplored. Contributing in this area can be a valuable contribution to the field.
3. **Comment Generation within Methods:** Generating comments for each line of code poses challenges. However, if accomplished, it can greatly benefit the IT industry by making code comprehension more efficient, potentially saving developers up to 30% of their time.

Despite advancements, the accuracy and results of comment generation models are still suboptimal. Achieving high accuracy in any of the aforementioned levels can pave the way for other researchers to explore DLT-based approaches in the same context.

## REFERENCES

1. Zheng Li, “SeCNN: A semantic CNN parser for code comment generation.” *Journal of Systems and Software* Volume 181, November 2021, 111036
2. Yu Zhou, “Augmenting Java method comments generation with context information based on neural networks” *Journal of Systems and Software* Volume 156, October 2019
3. Yuding Liang, “Automatic Generation of Text Descriptive Comments for Code Blocks”, *The Thirty-Second AAAI Conference on Artificial Intelligence (AAAI-18)*
4. Boao Li, “DeepCommenter: A Deep Code Comment Generation Tool with Hybrid Lexical and Syntactical Information”, *In Proceedings of the 28th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE '20), November 8–13, 2020, Virtual Event, USA. ACM, New York, NY, USA, 5 pages*
5. Yao Wan, “Improving Automatic Source Code Summarization via Deep Reinforcement Learning”, *In Proceedings of the 2018 33rd ACM/IEEE International Conference on Automated Software Engineering (ASE'18), September 3–7, 2018, Montpellier, France. ACM, New York, NY, USA, 11 pages.*
6. YunSeok Choi.” Source Code Summarization Using Attention-based Keyword Memory Networks” *2020 IEEE International Conference on Big Data and Smart Computing (BigComp)*
7. BolinWei. “Retrieve and Refine: Exemplar-based Neural Comment Generation” *In 35th IEEE/ACM International Conference on Automated Software Engineering (ASE '20), September 21–25, 2020, Virtual Event, Australia. ACM, New York, NY, USA, 12 pages.*
8. Alexander LeClair,” Improved Code Summarization via a Graph Neural Network”, *In Proceedings of ACM Conference, Washington, DC, USA, July 2017 (Conference'17),12 pages*
9. Miltiadis Allamanis,” A Convolutional Attention Network for Extreme Summarization of Source Code”, *arXiv:1602.03001*
10. Chen Lin,” Improving Code Summarization with Block-wise Abstract Syntax Tree Splitting”, *29th IEEE/ACM International Conference on Program Comprehension (ICPC 2021)*
11. Wasi Uddin Ahmad,” A Transformer-based Approach for Source Code Summarization”, *ACL 2020 virtual conference*
12. Wenhua Wang,” Reinforcement-Learning-Guided Source Code Summarization using Hierarchical Attention”, *IEEE Transactions on Software Engineering · March 2020*

13. DEZE WANG, “Deep Code-Comment Understanding and Assessment”, *IEEE Access (Volume: 7)*
14. Jian Zhang,” Automatically Generating Commit Messages from Diffs using Neural Machine Translation”, *IEEE/ACM 41st International Conference on Software Engineering (ICSE)*
15. Uri Alon,” CODE2SEQ: Generating sequences from structured representations of code”, *ICLR'2019*
16. Patrick Fernandes,” STRUCTURED NEURAL SUMMARIZATION”, *ICLR 2019*
17. Xing Hu,” Summarizing Source Code with Transferred API Knowledge”, *In Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence*
18. Xing Hu,” Deep Code Comment Generation”, *In Proceedings of IEEE/ACM International Conference on Program Comprehension, Gothenburg, Sweden, May 27 - May 28, 2018 (ICPC'18). ACM, New York, NY, USA, 11 pages.*
19. Alexander LeClair,” A Neural Model for Generating Natural Language Summaries of Program Subroutines”, *ICSE '19: Proceedings of the 41st International Conference on Software Engineering*
20. Bolin Wei,” Code Generation as a Dual Task of Code Summarization”, *33rd Conference on Neural Information Processing Systems (NeurIPS 2019), Vancouver, Canada*
21. B. Kitchenham and S. Charters, “Guidelines for performing systematic literature reviews in software engineering,” 2007.
22. “What is LSTM? Introduction to Long Short-Term Memory”, <https://intellipaat.com/blog/what-is-lstm/>
23. Keith, “A Brief History of Deep Learning”, [https://www.dataversity.net/brief-history-deep-learning\\_2022](https://www.dataversity.net/brief-history-deep-learning_2022)
24. “A Simple Introduction to Sequence-to-Sequence Models”, <https://www.analyticsvidhya.com/blog/2020/08/a-simple-introduction-to-sequence-to-sequence-models/>
25. Zeyu Sun, “A grammar-based structural CNN decoder for Code Generation”, *arXiv:1811.06837v1 [cs.LG] 14 Nov 2018*
26. XIAOTAO SONG “A Survey of Automatic Generation of Source Code Comments: Algorithms and Techniques”, *10.1109/ACCESS.2019.2931579*

PAPER NAME

**s\_final.pdf**

WORD COUNT

**7380 Words**

CHARACTER COUNT

**42183 Characters**

PAGE COUNT

**36 Pages**

FILE SIZE

**1.3MB**

SUBMISSION DATE

**May 30, 2023 11:42 AM GMT+5:30**

REPORT DATE

**May 30, 2023 11:42 AM GMT+5:30****● 10% Overall Similarity**

The combined total of all matches, including overlapping sources, for each database.

- 4% Internet database
- 7% Publications database
- Crossref database
- Crossref Posted Content database
- 3% Submitted Works database

**● Excluded from Similarity Report**

- Bibliographic material
- Quoted material
- Cited material
- Small Matches (Less than 8 words)

## List of Publications

### Priya Singh, Saransh Sharma, “A Literature Review on Automatic Comment Generation using Deep Learning Techniques”

**Published in:** 2022 3rd International Conference on Issues and Challenges in Intelligent Computing Techniques (ICICT) – Scopus

**Date of Conference:** 11-12 November 2022

**Date Added to IEEE Xplore:** 20 March 2023

Conferences > 2022 3rd International Confer... ?

### A Literature Review on Automatic Comment Generation using Deep Learning Techniques

Publisher: IEEE

Cite This

PDF

Priya Singh ; Saransh Sharma All Authors

24

Full

Text Views



#### Abstract

##### Document Sections

I. Introduction

II. Research Methodology

III. Related Work

IV. Results

V. Conclusion

Show Full Outline ▾

Authors

Figures

References

Keywords

Metrics

#### Abstract:

During the development of software, thousands of lines of codes are written and without proper comments it gets difficult for the developers to read the code. Automatic Comment Generation overcomes this problem by generating comments for the source code. Many researchers are already working on automatic comment generation in the source code and some have already published their results but still there is lot of scope of improvement in the field of automatic generation of comments in the source code. In this paper, we will be explaining and reviewing deep learning techniques that are used in comment generation to make the code easily readable. Generating comments automatically become crucial in the source code for any developer to understand it easily without spending much time. Hence, it's very important to understand how deep learning techniques like CNN, RNN, LSTM and others are making it possible to generate automatic comments in source code and also why researchers have now shifted on deep learning techniques to do automatic comment generation.

**Published in:** 2022 3rd International Conference on Issues and Challenges in Intelligent Computing Techniques (ICICT)

**Date of Conference:** 11-12 November 2022

**INSPEC Accession Number:** 22817404

**Date Added to IEEE Xplore:** 20 March 2023

**DOI:** 10.1109/ICICT55121.2022.10064537

► **ISBN Information:**

**Publisher:** IEEE

**Conference Location:** Ghaziabad, India

# PROOF:

Manuscript submission date extended till 03 September 2022

**Accepted papers will be submitted for the possible inclusion into IEEE Xplore.**

### WELCOME TO ICICT-2022

Department of Information Technology, KIET Group of Institutions, Ghaziabad, India in association with IEEE UP Section is organizing 3<sup>rd</sup> International Conference on Issues and Challenges in Intelligent Computing Techniques (ICICT-2022) on November 11-12, 2022. ICICT-2022 is technically co-sponsored by IEEE UP Section and IEEE USA. All accepted and presented papers will be submitted to IEEE for inclusion in IEEE Xplore Digital Library.

The main objective of the Conference is to stimulate and facilitate active exchange, interaction and comparison of approaches, methods and ideas related to specific topics, both theoretical and applied, in the general areas related to the Intelligent Computing, Communication, intelligent techniques, computing technologies, Software Engineering and other contemporary issues like High Performance Computing, Distributed Computing and Grid Computing to foster the exchange of concepts and ideas. The main aim of this International Conference is to contribute to academic arena, business world, and industrial community and in turn to the society.

Targeted audience of this conference would be representatives from Academia, Industry and Government Organizations who are involved or have interest in Computing Technologies and its Applications. Overall the conference will provide the researchers and attendees with prospects for national and international collaboration and networking among universities and institutions from India and Abroad for promoting research.

Conference proceedings that meet IEEE quality review standards may be eligible for inclusion in the IEEE Xplore Digital Library. IEEE reserves the right not to publish any proceedings that do not meet these standards. Conference related communications such as conference website, email, direct mail solicitation, etc. may not reference or guarantee inclusion in the IEEE Xplore Digital Library. The objective of the conference is to bring together researchers in the academic institutions and industries working in the related fields.

Indexing: The series will be submitted for inclusion to the leading indexing services including ISI Proceedings, EI-Compendex, DBLP, SCOPUS, Google Scholar and Springerlink.

**Paper Submission Link:** [Easy Chair Submission Portal](#)