DUPLICATE BUG REPORT DETECTION USING DATA AUGMENTATION TECHNIQUE

By

ANISA TARIQ

NATIONAL UNIVERSITY OF MODERN LANGUAGES

ISLAMABAD

December, 2025

DUPLICATE BUG REPORT DETECTION USING DATA AUGMENTATION TECHNIQUE

By

Anisa Tariq

Mcs, Virtual University of Pakistan, Lahore, 2018

A THESIS SUBMITTED IN PARTIAL

FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

MASTER OF SCIENCE

In Software Engineering

To

FACULTY OF ENGINEERING & COMPUTING

NATIONAL UNIVERSITY OF MODERN LANGUAGES ISLAMABAD

NATIONAL UNIUVERSITY OF MODERN LANGUAGES FACULTY OF ENGINEERIING & COMPUTING

THESIS AND DEFENSE APPROVAL FORM

The undersigned certify that they have read the following thesis, examined the defense, are satisfied with overall exam performance, and recommend the thesis to the Faculty of Engineering and Computer Sciences for acceptance.

Thesis Title: DUPLICATE BUG REPORT DETECTION USING DATA AUGMENTATION TECHNIQUE

Submitted by: Anisa Tariq **Registration #:** 59 MS/SE/S22

Master of Science in Software Engineering
Degree name in full

Software Engineering________
Discipline

Dr. Jaweria Kanwal________ Research Supervisor	____________________________ Signature of Research Supervisor
Dr. Sumaira Nazir__________ HOD (SE)	____________________________ Signature of HOD
Dr. Muhammad Noman Malik__ Dean (FE&C)	____________________________ Signature of Dean (FE&C)

December 11th, 2025

AUTHOR'S DECLARATION

I Anisa Tariq

Daughter of Malik Tariq Mehmood

Registration # 59 MS/SE/S22

Discipline Software Engineering

Candidate of **Master of Science in Software Engineering (MSSE)** at the National

University of Modern Languages do hereby declare that the thesis **DUPLICATE BUG REPORT DETECTION USING DATA AUGMENTATION TECHNIQUE** submitted by me in partial fulfillment of MSSE degree is my original work, and has not been submitted or published earlier. I also solemnly declare that it shall not, in the future, be submitted by me for obtaining any other degree from this or any other university or institution. I also understand that if evidence of plagiarism is found in my thesis/dissertation at any stage, even after the award of a degree, the work may be canceled and the degree revoked.

____
Signature of Candidate

ANISA TARIQ__
Name of Candidate

December, 2025__
Date

ABSTRACT

DUPLICATE BUG REPORT DETECTION USING DATA AUGMENTATION TECHNIQUE

In software projects, developers, testers, and end users identify bugs and report the bugs to the triager. To manage these bug reports, various Bug Tracking Systems (BTS) such as Bugzilla or Jira are used. One bug may be reported by multiple persons to the system which generate Duplicate Bug Reports in the system. Duplicate Bug Report Detection (DBRD) is very important because it results in a significant depletion of human resources. Many researchers proposed a range of machine learning techniques to detect the duplicate bug reports. The existing techniques performs well when a large number of bug reports are used as a training dataset but the performance of existing techniques significantly decreased for small dataset. To overcome this challenge, the data augmentation technique is used to increase the number of bug reports for the projects having small number of bug reports as training data. Various data augmentation techniques like synonym replacement, random insertion, component shuffling and class balance are used to increase the bug data. To validate the performance of data augmentation technique for duplicate bug detection, we used various deep learning models e.g. CNN, LSTM and BERT. We also compare the results of various deep learning techniques to analyze which model performs better with the augmented bug reports data. Our results show that data augmentation improved the results for all three models in term of accuracy, precision, recall, F1-socore and AUC score. The accuracy achieved on augmented data is 94.77%, 94.77% and 96.30% for LSTM, CNN and BERT respectively.

TABLE OF CONTENTS

CHAPTER	TITLE	PAGE
	ABSTRACT	**v**
	LIST OF FIGURES	**x**
	LIST OF TABLES	**xiii**
	LIST OF ABBREVIATIONS	**xiv**
	ACKNOWLEDGEMENT	**xv**
	DEDICATION	**xvi**
1.	**INTRODUCTION**	**1**
1.1	Introduction	1
1.2	Research Background	2
1.3	Problem Statement	3
1.4	Research Question	3
1.5	Research Objective	4
1.6	Motivation	4
1.7	Scope of Research	5
1.8	Research Contribution and Significance	5
1.9	Thesis Structure	6
1.10	Summary	6
2.	**LITERATURE REVIEW**	**7**

2.1 Introduction 7

2.2 Bug Report 7

2.3 Bug Life Cycle and Bug Tracking System (BTS) 8

2.4 Duplicate Bug Report 9

2.5 Natural Language Processing (NLP) 10

2.6 Natural Language Preprocessing Pipeline 10

2.6.1 Data Collection 11

2.6.2 Data Extraction and Cleaning 11

2.6.3 Data Preprocessing 12

2.7 Feature Engineering 13

2.8 Model Building 13

2.9 Model Evaluation 13

2.10 Machine Learning (ML) 15

2.10.1 Supervised learning 15

2.10.2 Unsupervised learning 16

2.10.3 Semi-Supervised learning 16

2.10.4 Reinforcement learning 16

2.11 Textual Features Extraction techniques 17

2.11.1 Bag of words (BOW) 17

2.11.2 Term Frequency-Inverse Document Frequency (TF-IDF) 17

2.11.3 Words Embedding 17

2.11.4 Context aware (BERT embedding) 18

2.12 Deep Learning (DL) Models 18

2.12.1 Convolutional Neural Network (CNN) 19

2.12.2 Recurrent Neural Network (RNN) 20

2.12.3 Long Short-Term Memory (LSTM) Networks 21

2.12.4 BERT Classifier Model: 23

2.13 Data Augmentation (DA) and techniques 27

2.13.1 Easy Data Augmentation 27

2.13.2 Back Translation 27

2.14 Data Augmentation Tool 27

2.14.1 Text Attack 28

2.15 Existing studies for duplicate bug report detection 29

2.16 Summary 34

3. RESEARCH METHODOLOGY 35

3.1 Data Collection 36

3.2 Targeted Features from dataset 37

3.3 Data Preprocessing 38

3.3.1 Loading Dataset 38

3.3.2 Data Cleaning 38

3.3.3 Tokenization 38

3.3.4 Feature Engineering 38

3.4 Data Augmentation 39

3.4.1 Synonym Replacement 40

3.4.2 Component Shuffling 41

3.4.3 Random Insertion 42

3.4.4 Class Balancing Strategy 42

3.4.5 Augmentation Rotation Strategy 43

3.4.6 Main Augmentation Interface 44

3.4.7 Data Generation Process 44

3.5 Model Selection 47

3.6 Training and Testing 48

3.7 Model Evaluation 49

3.8 Comparative analysis 49

3.9 Summary 50

4. RESULT AND DISCUSSION 51

4.1 Overview 51

4.2 Deep learning model results on Non-Augmented Data 52

4.2.1 LSTM Model 52

4.2.2 CNN Model 54

4.2.3 BERT Classifier 57

4.3 Deep learning model results on Augmented Data 59

4.3.1 LSTM Model 59

4.3.2 CNN Model 62

4.3.3 BERT Model 64

4.4 Comparison of Result, Augmented and Non-Augmented 67

4.4.1 LSTM Model 68

4.4.2 CNN Model 69

4.4.3 BERT Model 70

4.5 Result comparison with existing studies 71

4.6 Discussion 72

4.7 Summary 75

5. CONCLUSION AND FUTURE WORK **77**

5.1 Conclusion 77

5.2 Future Work 78

REFERENCES **79**

LIST OF FIGURES

FIGURE NO	TITLE	PAGE
Figure 2.1:	Bug Report Format [9]	8
Figure 2.2:	Bug life Cycle [11]	9
Figure 2.3:	Duplicate Bug Report [13]	10
Figure 2.4:	NLP Pipeline	11
Figure 2.5:	Working in ML [25]	15
Figure 2.6:	Working in DL [25]	19
Figure 2.7:	Convolutional Neural Network Architecture [34]	20
Figure 2.8:	Recurrent Neural Network architecture [37]	21
Figure 2.9:	LSTM Network Architecture [39]	22
Figure 2.10:	BERT architecture [44]	24
Figure 2.11:	BERT architecture layers [47]	25
Figure 2.12:	Methods of Data Augmentation [52]	28
Figure 3.1:	Research Methodology and Experimental Procedure	36
Figure 3.2:	Data set column	37
Figure 3.3:	Working of BERT Classifier	48
Figure 4.1:	LSTM Accuracy and Loss	53
Figure 4.2:	LSTM ROC Curve	53
Figure 4.3:	LSTM Confusion Matrix	54
Figure 4.4:	CNN Training vs Validation Loss	55

Figure 4.5:	CNN Training vs Validation Accuracy	55
Figure 4.6:	CNN ROC Curve	56
Figure 4.7:	CNN Confusion Matrix	56
Figure 4.8:	BERT Training vs Validation Loss and Accuracy	57
Figure 4.9:	BERT ROC Curve	58
Figure 4.10:	BERT Confusion Matrix	58
Figure 4.11:	LSTM model training vs Validation Accuracy and Loss	60
Figure 4.12:	LSTM ROC Curve	61
Figure 4.13:	LSTM Confusion Matrix	61
Figure 4.14:	CNN model training vs Validation Accuracy and Loss	62
Figure 4.15:	CNN ROC Curve	63
Figure 4.16:	CNN Confusion Matrix	63
Figure 4.17:	BERT Training vs Validation Accuracy	64
Figure 4.18:	BERT Training vs Validation Loss	65
Figure 4.19:	BERT ROC Curve	65
Figure 4.20:	BERT Confusion Matrix	66

LIST OF TABLES

TABLE NO	TITLE	PAGE
Table 3.1	Data set	37
Table 4.1	Result of non-augmented data	52
Table 4.2	Result of augmented data	59
Table 4.3	Comparison of results	68
Table 4.4	Result comparison with existing studies	71

LIST OF ABBREVIATIONS

BR	Bug Report
DBRD	Duplicate Bug Report Detection
BTS	Bug Tracking System
NLP	Natural Language Process
ML	Machine Learning
DL	Deep Learning
CNN	Convolutional Neural Network
RNN	Recurrent Neural Network
LSTM	Long Short Term Memory
BERT	Bidirectional Encoder Representations from Transformers
EDA	Easy Data Augmentation
TF-IDF	Term frequency-Inverse document frequency

ACKNOWLEDGEMENT

I would like to express my profound gratitude to Almighty Allah for His divine guidance and blessings, which have made this research endeavor possible.

I extend my sincere appreciation to all individuals and organizations that have significantly contributed to the success of this study. My heartfelt thanks go to my research supervisor, whose invaluable guidance and unwavering support have profoundly shaped my research journey. I am also grateful to the Department of Software Engineering for providing a conducive research environment and to the administrative staff for their efficient support throughout this process.

Lastly, I want to thank everyone whose names may not be mentioned here but who has played a role in my success and personal growth. Your efforts have not gone unnoticed, and I am truly grateful.

DEDICATION

This research is dedicated to the pillars of my life: my parents, who selflessly supported and encouraged me to pursue my dreams, my siblings, who were by my side through every up and down; and my teachers, who instilled in me the knowledge and values that shaped my academic journey. I am eternally grateful for their unwavering love, guidance, and motivation, which have been the driving force behind my success. Thank you all for boosting me to get up and keep going. Lastly, I extend my gratitude to all those individuals who in any way contribute to the success of this endeavor. "May Allah bless them all”

CHAPTER 1

INTRODUCTION

1.1 Introduction

In software development and maintaining, bugs are reported by the developers, testers and users. Managing bug reports effectively is essential for maintaining the quality and reliability of a software. In open source project, bug repositories are used to keep the record of bugs. These bug repositories are called bug tracking systems (BTS).There are following bug tracking systems like Bugzilla, Jira, Redmine and GitHub issue tacking etc. Bug tracking systems collect reports from users and testers to help identify and fix issues in the software. Developer and users from all over the world can submit the bug report. Using bug tracking software, we make the record of the bugs and easily it can be addressed.

Bug reporting is a critical component of developing and maintaining software. Bug report is detail document that explains the bug, its severity, priority, version, title, summary and description. Bug report is called duplicate bug report (BDR) when same bug is addressed by multiple persons and submit report in BTS [1]. Detecting duplicate bug reports is a challenging task. Recent data from the Eclipse Bug Tracking System (BTS) shows that, over three years (January 1, 2018, to January 1, 2021), 7542 bug reports were submitted. Approximately three-quarters of these reports were either erroneous or duplicates [2]. So, it is essential to improve mechanisms for finding bug reports similar or redundant. Handling these almost unavoidable frequent bug reports requires considerable maintenance work and prioritization and resolution [3]. We can utilize the existing information to fix bugs by efficiently identifying duplicate bug reports. [4].

Bug report is written in natural language text. Developer, tester and user write same bug report using different vocabulary that cause duplicate bug report. Bug reports use semantic textual similarity (STS) techniques, which compare text similarities based on semantic meaning. Machine learning methods and natural language processing (NLP) methods have been applied for the same purpose.

Duplicate bug report has been identified in several researchers by recognizing high textual similarity between information (such as bug report titles and descriptions) [2]. In order to prevent needing to repair bugs again, these duplicates must be found. However, the daily bug report management process is typically effortful task that becomes more difficult as the project goes on. The project can be handled with ease if it is automatically identified that a bug report is duplicate. Over the past few years, numerous Duplicate Bug Report Detection (DBRD) techniques have been introduced.

Identification and elimination of these duplicates assist in enhancing the debugging process. This research focuses on using data augmentation to improve the accuracy of machine learning models in identifying duplicate bug reports, particularly when dealing with small datasets.

1.2 Research Background

In duplicate bug report identification, current methods such as conventional Information Retrieval (IR) techniques and state-of-the-art deep learning (DL) models have turned out to be suboptimal because of numerous reasons. Studies have highlighted that these methods tend to lose some of the semantic similarity between bug reports. It causes the main reason in losing important information. Many machine learning and deep learning models do not perform well when they are trained on small amount of datasets. These model needs a large amount of data to learn patterns effectively. These models can learn better and provide better results when there is sufficient data available for training. Some new or small scale software projects have less number of historical bug reports. So this limited amount of data limits the ability of machine learning and deep learning algorithms to learn important patterns and it makes difficult to predict duplicate reports accurately.

There is one solution to overcome the challenge of limited data set. We can increase the size of the dataset through data augmentation methods. Data augmentation generates new data set with diversity that can be used for training machine learning models. It can help in enhancing the learning process and accuracy of the model.

1.3 Problem Statement

Machine learning and deep learning techniques perform well and achieve high accuracy when trained on large and diverse datasets. It struggles with predicting accurate results when trained on small datasets [3] [4]. Similarly software project with limited historical bug reports make it difficult for model to achieve high accuracy in detecting duplicate bug reports (BRs) [5].

Researchers explored different machine learning and deep learning techniques such as random forest, decision tree, SVM, CNN, LSTM and transformer based models like BERT [6]. These models performed well in identifying duplicate bug reports with large number of historical data.

To address the given challenge, our focus on increase large amount of training data. For this purpose, data augmentation technique can be applied to synthetically increase the dataset and improve the models performance.

1.4 Research Question

There is only one research question related given problem statement.

RQ: How does data augmentation affect the accuracy of machine learning techniques for small datasets in detecting duplicate bug report?

1.5 Research Objective

Obj: To evaluate the impact of augmentation techniques on the accuracy of machine learning models in duplicate bug report detection when bug reports data is limited.

1.6 Motivation

The motivation behind this topic “Duplicate Bug Report Detection using Data Augmentation Technique” arises from the growing the challenge to identify the duplicate bug report in bug tracking system when the available data is limited. In many software projects, there is less number of bug reports and especially duplicate ones. This limited amount of data poses a significant obstacle to effectively training machine learning models, which typically require large volumes of labeled data to learn meaningful patterns and achieve high accuracy. When machine learning models are trained on limited data set, it reduces the performance of models and it gives inaccurate outcomes in identifying duplicate bug reports.

To overcome this challenge, data augmentation technique gives a practical solution. Augmentation technique is a novel technique that artificially increase the data set using different strategies. So data set can be increased using data augmentation techniques. It provides more data for a machine learning model for training, which helps in increasing the machine learning model performance in terms of accuracy. When we have small number of bug reports in bug tracking system so it is not helpful for machine learning models to train the enough data for better results. By increasing the size and richness of the dataset, data augmentation enables machine learning and deep learning models to generalize better, identify subtle semantic similarities and improve accuracy in detecting duplicates.

As we know that there are different machine learning models that produce results according to their unique mechanism. So there is motivation to compare the results of different machine learning model that how they improve their results when trained with augmented data. We decided to compare the results of three machine learning Models LSTM, CNN and BERT because these models performed well or large data set according existing literature.

1.7 Scope of Research

The purpose of this research is to address the challenges posed by limited bug reports in bug tracking system and also to explore the impact of the data augmentation technique on the performance of machine learning and deep learning models in identifying duplicate bug reports when it is trained on new artificially increased data set. The research process includes some main steps: data collection, preprocessing, artificial expansion of the dataset through augmentation, model training, testing, and performance evaluation. Since small datasets are usually inadequate to train good models so data augmentation is applied to synthetically boost the size and diversity of training data. This research particularly addresses the impact of augmentation methods for text data. Since bug reports are typically written in natural language, this study emphasizes the application of augmentation strategies that preserve semantic meaning while increasing data volume and variety.

1.8 Research Contribution and Significance

This research makes a significant contribution. This study informs us about the significance of data. A limited amount of data does not assist the machine learning models in predicting accurate and reliable results. To achieve this purpose we employ data augmentation technique which assist in expanding the amount of training data. Data augmentation technique gives sufficient and diverse data to train the machine learning and deep learning models which enables them to learn better pattern and make more accurate prediction. It improves the accuracy of machine learning model. By improving model performance, data augmentation directly contributes to more effective duplicate bug report detection, which is critical for maintaining software quality. It helps development teams reduce time spent on redundant bug handling and improves overall efficiency. Ultimately this research will help to improve the project management task.

1.9 Thesis Structure

Thesis is organized in following chapters that describes about bug, bug report and methods of detecting duplicate bug report. Chapter 1 is dedicated to introduction of the thesis. It outlines the problem statement. Research question and objectives. It also outlines the motivation and study scope for the topic of thesis. Chapter 2 is comprehensive background information and review of literature of bug, bug report, bug tracking system, natural language processing, machine learning, deep learning and augmentation techniques. It provides information about techniques that has been utilized for finding duplicate bug report. It also tells about augmentation techniques that how efficiently enhances the data set size and aids in training. Chapter 3 is research methodology. It explains the steps of data collection, data preprocessing, and data augmentation process, training of the data set, testing and evaluation process and performance evaluation metrics. Chapter 4 is for result discussion and analysis. It describes the result as well as the comparison of results of both non-augmented and data augmented. It discusses that how augmentation technique has enhanced the accuracy of the machine leaning and deep learning model. Chapter 5 discusses thesis work conclusion and give directions regarding the future work that will be useful for future researchers.

1.10 Summary

This chapter defines the problem statement that limited number of bug reports does not provide enough data for machine learning model to accurately identify duplicate bug reports. To address this specific problem we use a novel techniques that is data augmentation. Data augmentation methods assist in enhancing the synthetically data in terms of size that assist machine learning models to learn more effectively and predict the outcome more accurately. This technique enhances model’s performance. It also highlights the key component including research question, research objective, scope, and contributions. These elements help in preparing foundation for the chapter that follows.

CHAPTER 2

LITERATURE REVIEW

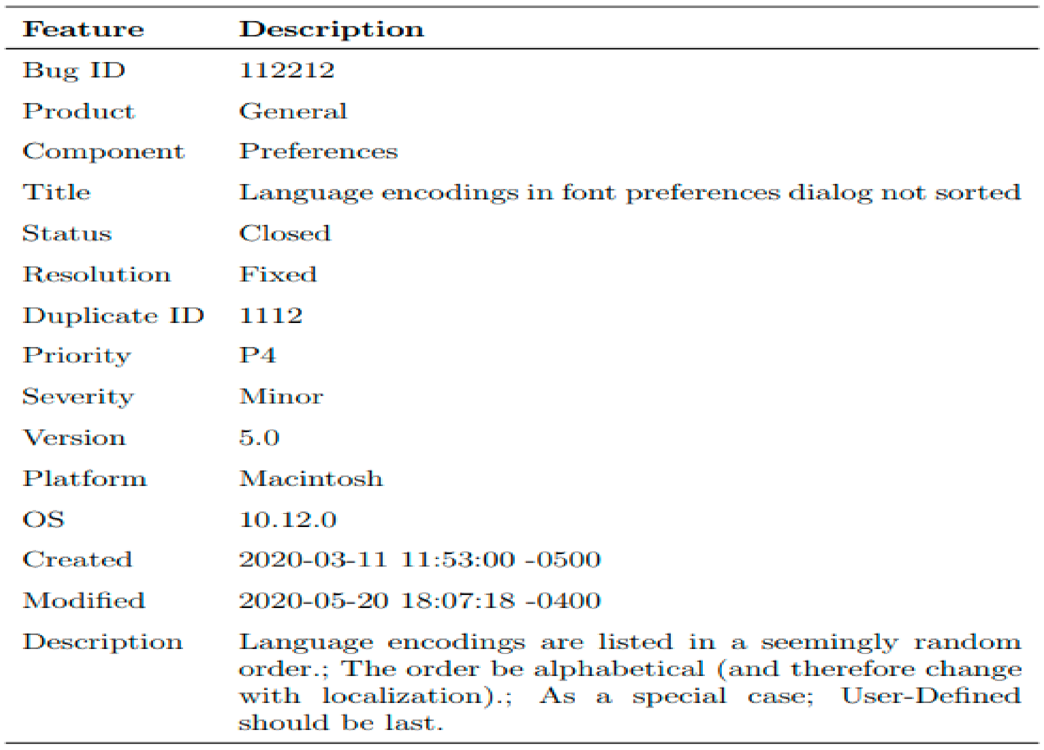
2.1 Introduction

This chapter provides a detailed overview of existing studies related to bug, bug life cycle, bug report, duplicate bug report. It also provides existing knowledge about duplicate bug report detection techniques and various data augmentation techniques applicable to textual data. It also provides comprehensive study about natural language processing (NLP), its concept, techniques and working. This chapter also covers machine learning and deep learning techniques and their architecture, highlighting how they are applied in the context of bug report analysis. This chapter establishes a strong theoretical foundation that guides the research methodology and experimental design presented in later chapters.

2.2 Bug Report

A software developers make a software for users according to their requirements. He fulfils their requirements if requirements does not meet, it means there is an error. Developers write a code according to requirement of the programs. When there is human mistake or errors during writing a code that can affect the functionality of the program. It is called bug or defect. After finding the bug, User or tester write the bug report. The word “Bug” is formally used since 1870 [7].

After identifying the bug, Bug report is written. It is a document that contain different information about bug like Bug ID, Product, Component, Title, Status, Resolution, Duplicate Bug report ID, Priority, Severity, Created, Summary and description. Figure 2.1 explains the bug report. Bug report data is divided into structured and unstructured data. Structured data contains Bug ID, Product, Component, Status, Resolution, Duplicate ID, Priority, Severity, Version, and Platform. Structured data is categorical data. Unstructured data is Title, Summary and Description.[8]. It is textual data.



Feature	Description
Bug ID	112212
Product	General
Component	Preferences
Title	Language encodings in font preferences dialog not sorted
Status	Closed
Resolution	Fixed
Duplicate ID	1112
Priority	P4
Severity	Minor
Version	5.0
Platform	Macintosh
OS	10.12.0
Created	2020-03-11 11:53:00 -0500
Modified	2020-05-20 18:07:18 -0400
Description	Language encodings are listed in a seemingly random order.; The order be alphabetical (and therefore change with localization).; As a special case; User-Defined should be last.

Figure 2.1: Bug Report Format [9]

2.3 Bug Life Cycle and Bug Tracking System (BTS)

Reporting bugs is an essential part of developing, testing, and maintaining software. Typically, a bug tracking system like Bugzilla, Jira and Redmine is used to submit and analyze reported bugs. A person called software triager, who analyze the bug report. He is knowledgeable about the project, system, and developers [10]. Bug is submitted in bug tracking system.

Bug has a life cycle that stats different information. When a bug is sent to the bug tracking system it called ‘new bug’ then it is assigned to the developers. Developer fix the bug. Sometimes it is not fixed for a few reasons; either the bug is invalid or the bug is already resolved, or it is a duplicate. Figure 2.2 explains the bug lifecycle.

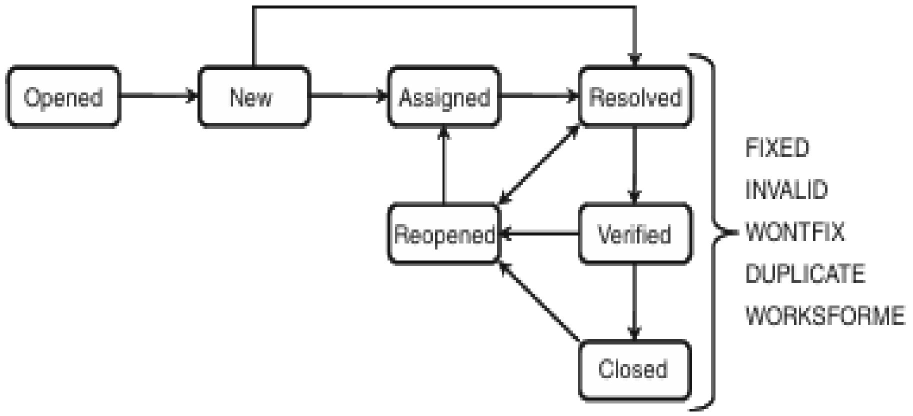
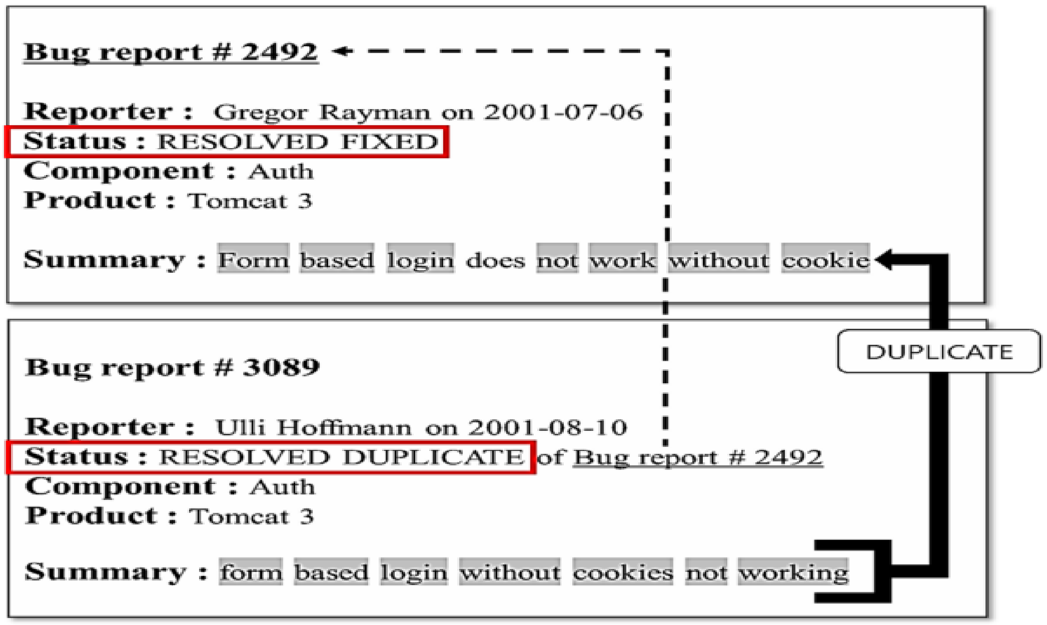


Figure 2.2: Bug life Cycle [11]

2.4 Duplicate Bug Report

Bug report is written in natural language text. Duplicate bug reports arise when multiple individuals file bug reports for the same issue [1]. Developer, tester and user write same bug report using different vocabulary that cause duplicate bug report. Bug reports involve semantic textual similarity (STS) techniques, which compare text on the basis of semantic similarities. Duplicate bug report has been identified in several researchers by recognizing high textual similarity between information (such as bug report titles and descriptions) [2]. Developer mark it as duplicates bug report if it has been fixed. In order to prevent needing to repair bugs again, these duplicates must be found.

Textual similarity is determined by extracting keywords and assessing how alike they are. These methods employ various techniques such as word embedding, TF-IDF and bag-of-words. Some other approaches depend on automatically condensing keywords as features for training binary classifiers, such as XGBoost [12].


Bug report # 2492

Reporter : Gregor Rayman on 2001-07-06
Status : RESOLVED FIXED
Component : Auth
Product : Tomcat 3

Summary : Form based login does not work without cookie

Bug report # 3089

Reporter : Ulli Hoffmann on 2001-08-10
Status : RESOLVED DUPLICATE of Bug report # 2492
Component : Auth
Product : Tomcat 3

Summary : form based login without cookies not working

Figure 2.3: Duplicate Bug Report [13]

2.5 Natural Language Processing (NLP)

Natural Language processing is the part of artificial intelligence. It processes the human language into computer language. It is used in different computer application like speech recognition, Chabot, sentiment analysis, machine translation and text classification. It fills the gap between machines and human. Different techniques of ML and DL are applied for natural language processing improvements [14].

2.6 Natural Language Preprocessing Pipeline

Natural language processing follows a systematic approach that convert human language into machine language and trains machine learning and deep learning model for real life applications. Figure 2.4 explains the whole NLP pipeline which explain the flow of working.

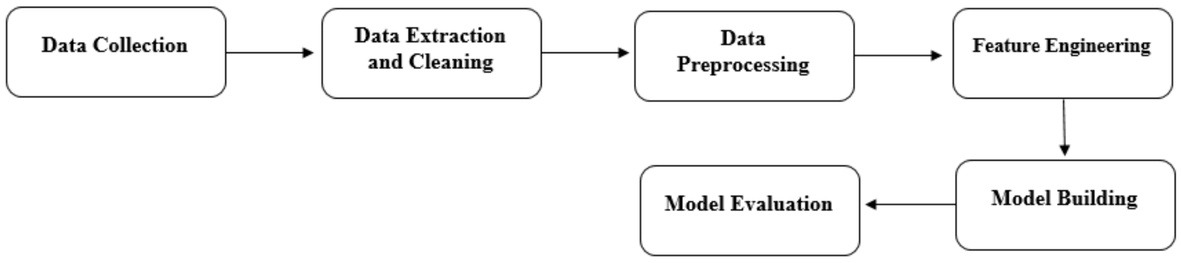


Figure 2.4: NLP Pipeline

2.6.1 Data Collection

The first step in an NLP pipeline is data collection. It is a crucial part of the process, as the quality and quantity of data directly impact the performance of the model. Various techniques can be used for data acquisition. For example, web scraping is a common method for collecting data from online sources. Surveys can also be conducted to gather relevant data. The choice of data collection technique depends on the specific topic or problem you are addressing. Therefore, it is important to obtain the necessary data in order to effectively solve the targeted problem.

2.6.2 Data Extraction and Cleaning

After collecting the dataset, the next step is to identify and keep only the relevant information. In most cases, the raw data is not well defined or well organized. Therefore, we need to extract the useful parts based on our specific requirements and discard any irrelevant or unnecessary information. We can create simple and better version of our data by removing irrelevant data. We can check spelling mistakes and we can remove punctuations, different html tags in data set and convert text in lower casing.

2.6.3 Data Preprocessing

In natural language processing (NLP), data is in form of text. Textual data is written in paragraphs, sentences or in lines. Computer cannot understand natural language or human language. Data preprocessing is necessary step that convert the unstructured text into structured format that machine learning model can understand it and learn effectively[15]. Different data preprocessing techniques are following there.

Tokenization

In NLP model, Tokenization in important step. Tokenization means you have to break the text data into small units. It provides better understanding for NLP models [16]. There are two types of tokenization. Sentence tokenization and word tokenization

Sentence Tokenization

In sentence tokenization we break the whole paragraph is sentences. This process is called sentence segmentation or sentence tokenization.

Word Tokenization

In word tokenization we break the sentence into words. So it is important process for NLP model. On the basis of word tokenization we can train machine learning models and can build NLP applications.

Stemming

Stemming is the process which extract the base word and remove the affix. Stemming does not follow grammar. It has fix rule to extract the word [17].

Lemmatization

Lemmatization is one step ahead from stemming. It extract meaningful base word following grammar rule. Stemming provides best result as compared to stemming because it understand the semantics of word [18][17].

2.7 Feature Engineering

As we know that machine learning models do not understand the textual data directly. They only understand numbers so for this purpose we use different techniques to convert textual data into numbers. These numbers are meaningful representation of textual data. These numbers represent vectors. This whole process is called feature engineering [19]. Different feature engineering models has been proposed like Term frequency and Inverse document frequency (TF-IDF), One Hot encoding and Words embedding Word2vec[20][19].

2.8 Model Building

After completing the feature engineering, next step is to build machine learning model. All machine learning and deep learning models are not useful for natural language processing but classification model is helpful in it. Different classification techniques are Random forest, Decision tree, Support Vector machine (SVM), Naïve Bayes, Convolutional neural network (CNN) and Recurrent convolutional network (RNN) and Transformers like BERT [21] [22]. Using any model we train data set and predict the outcomes.

2.9 Model Evaluation

Model evaluation is a process to check the efficiency of machine learning and deep learning models that how affectively models learn from training data set and predicts the accurate results. There are following evaluation metrics that assess the performance of machine and deep learning models. Confusion matrix is one of them that helps to evaluate the model performance. We use various evaluation metrics as well like accuracy, precision, recall, and f1-score to evaluate the model performance [23].

Accuracy

There are four predictions, true positive (TP), true negative (TN), false positive (FP) and false negative (FN). It predicts the ratio of correct or true prediction from total number of predictions. It can be true positive or true negative. It provides an overall result of the classifier is correct.

$$\frac{\mathbf{TP+TN}}{\mathbf{TP+TN+FP+FN}} \quad (1)$$

Precision

It measure that how many predictions are actually correct from all positive prediction either true positive or false positive.

$$\frac{\mathbf{TP}}{\mathbf{TP+FP}} \quad (2)$$

Recall

It predicts the true positive prediction from made up of all positive class.

$$\frac{\mathbf{TP}}{\mathbf{TP+FN}} \quad (3)$$

F1-score

It provides balance score that covers precision and recall both.

$$\frac{\mathbf{2(Precision*Recall)}}{\mathbf{Precision+Recall}} \quad (4)$$

ROC-AUC Curve

The Receiver Operating Characteristic (ROC) - The Area Under the Curve (AUC) is useful in binary classification tasks. It provides result for classification model performance.

2.10 Machine Learning (ML)

Machine learning is the branch of artificial intelligence (AI). It is used for data analysis, where it learns patterns and extracts features from the given datasets. These datasets are then trained using different algorithms. Machine learning models and techniques predict results and make decisions on the basis of training data set. Supervised learning, unsupervised learning, semi-supervised learning and reinforcement learning are the four categories of machine learning. [24]. Figure 2.5 shows the working of Machine learning model.

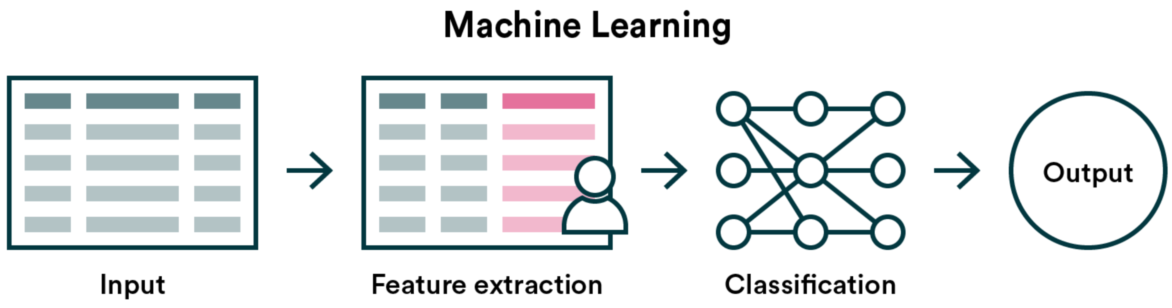


Figure 2.5: Working in ML [25]

2.10.1 Supervised learning

Supervised learning is very important in machine learning. It is used to train models on the labeled data set. Data is given as input to algorithm. It learns from input and gives output or predict results according the variable that we set [26]. Supervised learning is categorized in two types: Classification and regression. In regression we predict numerical values like temperature, score, stock prices etc. In classification we predict different things like either it's a cat or a dog. In NLP, tasks like text classification, sentiment analysis, and duplicate detection often rely on this approach. Each text sample is paired with a target label, such as a category or class. The model learns patterns in the labeled data and generalizes to unseen text. Supervised learning algorithms like Random Forest, Decision Trees, SVM, and deep neural networks are commonly used here.

2.10.2 Unsupervised learning

Unsupervised learning is done when there is unlabeled data. It means we give data as input but without any labeled outcome. Algorithms learns the pattern and predict the result by itself [26]. In NLP, it is often used for clustering similar documents, topic modeling, or word embedding learning. Models like K-Means or Latent Dirichlet Allocation (LDA) find structure in data without needing labels. This is useful for organizing large text corpora or exploring data before annotation. It helps in understanding semantic relationships between words or reports.

2.10.3 Semi-Supervised learning

Semi-supervised learning is done when we have labeled and unlabeled data. It improves the performance of model and provides better results [27]. Classification and clustering both are used in semi-supervised learning. This is especially helpful in NLP tasks where manual labeling is costly and time-consuming. The model initially learns from labeled data, then refines its learning using patterns from the unlabeled data. It is commonly applied in tasks like named entity recognition and duplicate detection when labels are limited. Techniques often include self-training, co-training, or graph-based methods.

2.10.4 Reinforcement learning

Reinforcement learning is done as making decision on the basis of trial and error [26]. It is used in complex dataset. In NLP, it's applied in tasks like dialogue systems, text summarization, and machine translation, where the system improves through trial and error. The model learns strategies for generating or selecting the best text output based on reward signals. Over time, it optimizes for long-term success in language interactions. Techniques like policy gradients and Q-learning are used in these scenarios.

2.11 Textual Features Extraction techniques

We use following techniques for feature extraction. It converts textual data into vectorization form.[28][19].

2.11.1 Bag of words (BOW)

Represents text as a collection of word counts without considering grammar or word order. It only works to count the words that appears frequently in document. It is not complicated to use but it does not work on semantics and words order. It cannot understand the relationship between words and often leads to high-dimensional sparse feature vectors [29].

2.11.2 Term Frequency-Inverse Document Frequency (TF-IDF)

TF-IDF works same like bag of words but it is improved version of BOW. It works by considering how important a word is to a document in a collection. TF counts how often a word appears in a document. IDF reduces the weight of common words across documents [29]. It does not give much importance to common words. It also not understand the semantics in text.

2.11.3 Words Embedding

Words embedding are a powerful textual feature extraction technique that represent words as dense, low-dimensional vectors that capture their semantic meaning. These embedding map similar words to similar vector spaces, enabling models to understand relationships between words beyond simple frequency counts [30]. Among the most widely used embedding methods are Word2Vec and GloVe**.**

GloVe

GloVe (Global Vectors for Word Representation), developed by Stanford, is based on matrix factorization and constructs word vectors by analyzing global word-word co-occurrence statistics across a corpus. Unlike Word2Vec, which is predictive, GloVe is count-based and captures more global context of words[31]. It covers semantics and count words how many time a similar words appears in document. But it is used when we have large number of data set.

Word2vec

Word2Vec is developed by Google, learns word representations using neural networks by predicting a word based on its surrounding context (Skip-gram) or predicting surrounding words given a central word (CBOW). It captures both syntactic and semantic relationships by training on large corpora and is known for producing meaningful vector arithmetic. It represents the similar word with semantics and syntactic. It understand the relationship between words in document [32]. It represents the same numerical form for the word having same meaning. But one problem is that it cannot handle polysemy.

2.11.4 Context aware (BERT embedding)

BERT is pre-trained using masked language modeling**.** It works same like glove and word2vec but it produces contextual embedding**,** meaning the same word can have different vectors depending on its sentence context. It uses a transformer architecture that considers the entire sentence bi-directionally, capturing deeper semantic relationships [22]. It is also helpful for large and complex data. It requires more storage and computation power that is its limitation.

2.12 Deep Learning (DL) Models

Machine learning models require features as input to learn and make predictions. In contrast, deep learning is an advanced version of machine learning that draws inspiration from the human brain and can automatically learn features from raw data using multiple layers. Deep learning model learns features by themselves and predict results. Deep learning is high efficient for complex data and high dimensional data sets [33].

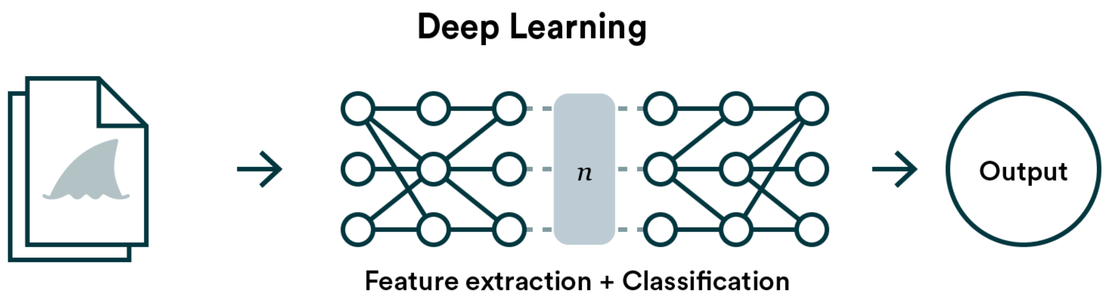


Figure 2.6: Working in DL [25]

Deep learning models have achieved immense success in Natural Language Processing (NLP) in recent years. Deep learning models have the ability to understand the semantics in NLP from raw text without manual feature engineering. In deep learning models, Convolutional Neural Networks (CNNs) and Long Short-Term Memory (LSTM) networks are the powerful model. They have shown impressive ability in text classification, similarity detection, and semantic matching. This makes them especially effective for identifying duplicate bug reports.

2.12.1 Convolutional Neural Network (CNN)

Convolutional neural network is a deep learning model that is used in computer vision like image identification and natural language processing like text classification. It is used in NLP for sentiment analysis and document categorization. It achieved very promising results in computer vision and NLP. It is very affective due to its architecture and way of working. The architecture of a CNN for NLP commences with an embedding layer that converts each word in a sentence into a dense vector representation. This whole process is followed by one or more convolutional layers that apply filters to extract local n-gram features from the text. Figure 2.7 explains the working architecture of CNN.

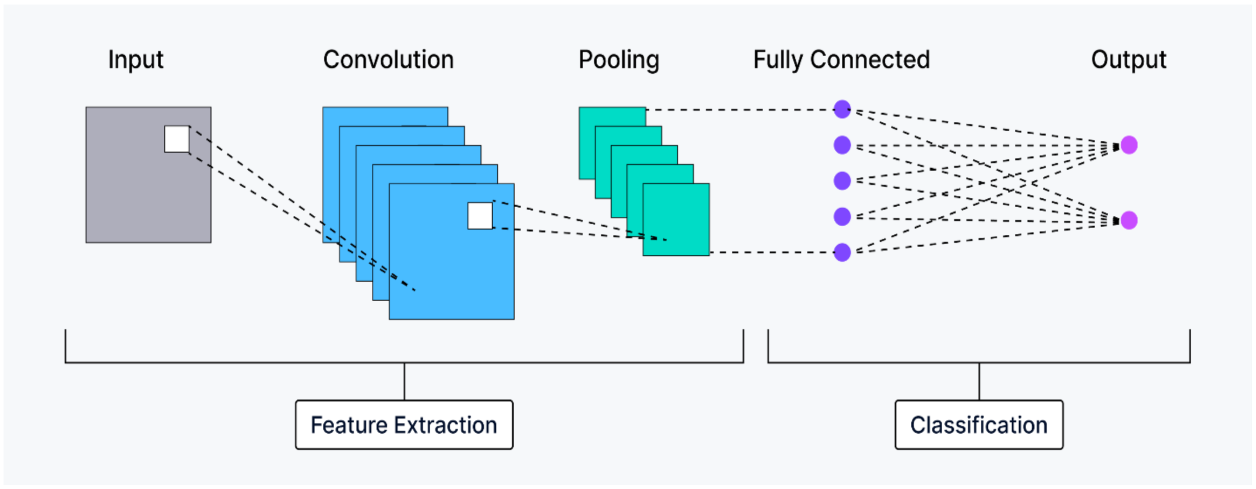


Figure 2.7: Convolutional Neural Network Architecture [34]

These convolutional layers are good enough at capturing position invariant patterns and local dependencies in text, such as frequent keyword combinations. The activation function, usually ReLU is applied. It introduces non-linearity to the model and helps it to learn complex relationships. A max-pooling layer is then used to select the most prominent features by reducing the dimensionality and retaining the most meaningful information [35].

The output from the pooling layer is flattened and passed to fully connect (dense) layers, which further process the extracted features for the final classification task. The final layer mostly applies a softmax or sigmoid activation. When there is requirement either the output is multi class or binary classification. CNNs model can be trained on textual data to check the textual similarity between two bug reports in identifying duplicate bug report.

2.12.2 Recurrent Neural Network (RNN)

Recurrent Neural Network (RNN) is one of the neural networks that specifically designed to process sequential data**.** It is highly ideal for natural language processing (NLP) tasks. It works making a series or sequence. It generates results according to input sequences that is given to model. RNNs have multiple layer. There are some hidden layers that captures information about previous input. RNNs can also learn the timing and order of words in sequences like sentences or documents. This helps them understand how words are connected and what they mean together. RNNs are useful for tasks like language modeling, translating languages, sentiment analysis in text, and finding duplicate text or classification [36].

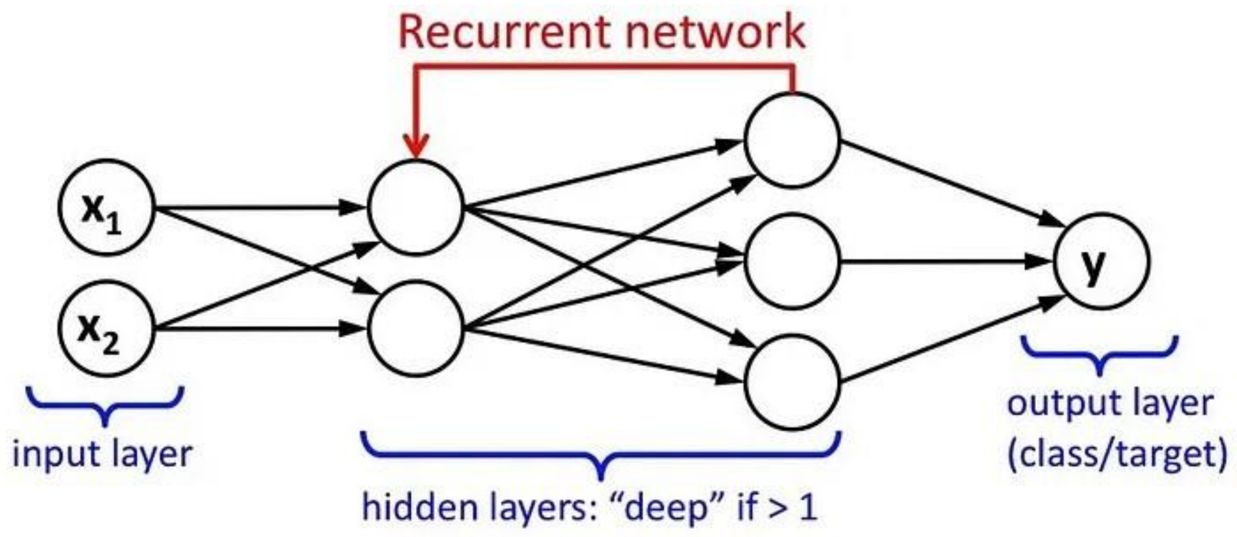


Figure 2.8: Recurrent Neural Network architecture [37]

RNNs are good at understanding both the structure and meaning of language. This makes them very useful for tasks like classifying text, labeling parts of a sequence, and answering questions. They can also be extended to Siamese architectures for tasks like sentence similarity and duplicate detection by comparing vector representations of text sequences learned by shared RNN branches.

2.12.3 Long Short-Term Memory (LSTM) Networks

LSTM is a type of recurrent neural network (RNN) that is specifically designed for sequential data. It performs very well on sequential data like RNNs. LSTM is ideal for NLP tasks because it can understand the word order or semantic similarity. Word order significantly impacts in meaning. LSTM networks address the shortcoming the limitations of traditional RNNs through integration memory cells and specialized gates like input, forget, and output gates that control how information is stored, used, or removed as the network learns [38].

An LSTM based architecture for duplicate detection begins with an embedding layer, followed by one or more LSTM layers that is used to process the input sequences word by word. The memory cells store information over long distances, capturing the semantic flow of sentences. This is crucial when comparing two bug reports that use different phrasing but convey the same issue.

The final hidden states from the LSTM layers can be either passed directly to a dense layer for classification or combined with attention mechanisms to enhance interpretability. This allows model to learn and focus on specific parts input sequence when deciding how similar things are. It is helpful for improving the performance in complex sentence structures.

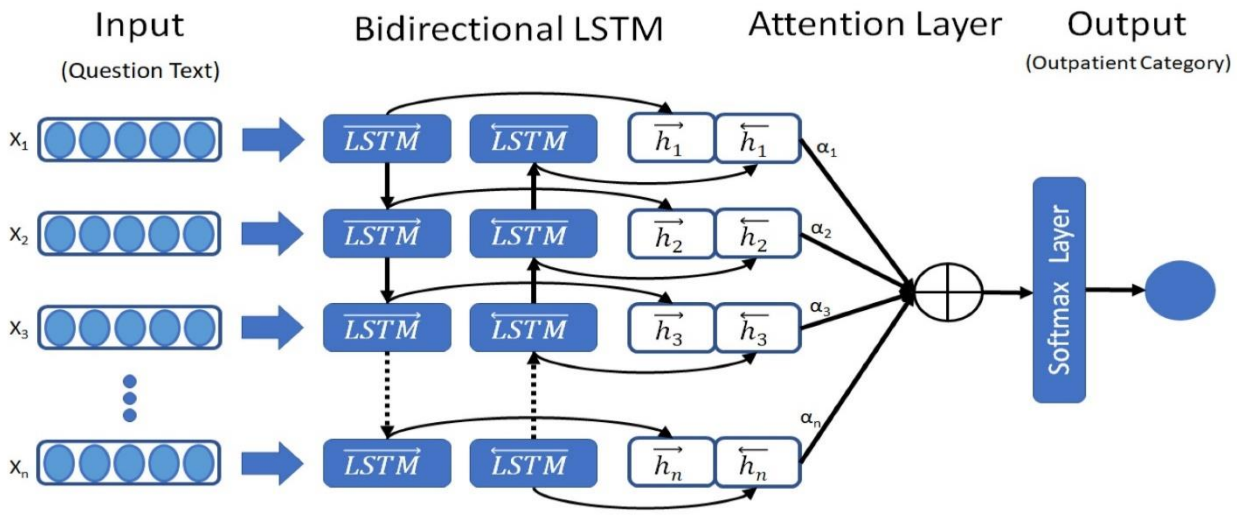


Figure 2.9: LSTM Network Architecture [39]

Transformer Models

Vaswani et al. introduced “The Transformer architecture” in 2017 in the paper “Attention is All You Need,” marked a revolutionary shift in natural language processing [40]. Deep learning and machine learning model are working on NLP. Like BERT, GPT and T5 [41].

2.12.4 BERT Classifier Model:

BERT (Bidirectional Encoder Representation from transformers) is a transformer base deep learning model that is introduced by Google in 2018 It had a huge impact and totally changed the landscape of Natural Language Processing (NLP) [42]. It has been trained on large amount of text data. That is why it is pre-trained model. It has the ability to understand the contextual meaning from text data including Wikipedia and books corpus. It has two key objectives: Masked Language model (MLM) and next sentence prediction (NSP)[43].

Masked Language Modeling (MLM): In this task, some percentage of the input tokens are randomly masked and the model learns to predict these masked tokens based on the context provided by the unmasked tokens. This forces the model to develop a contextual understanding of the sentence structure and word usage.

Next Sentence Prediction (NSP): The model is given pairs of sentences and must predict whether the second sentence logically follows the first. This helps BERT understand relationships between sentences.

BERT Variant: BERT-base and BERT-large are two versions of BERT. BERT-base has 12 encoded layers, 768 hidden units, 12 self-attention heads, and 110 million parameters. BERT-large has 24 encoded layers 1024 hidden units, 16 self-attention heads, and 340 million parameters. [44]. Fig explains the working of BERT classifier.

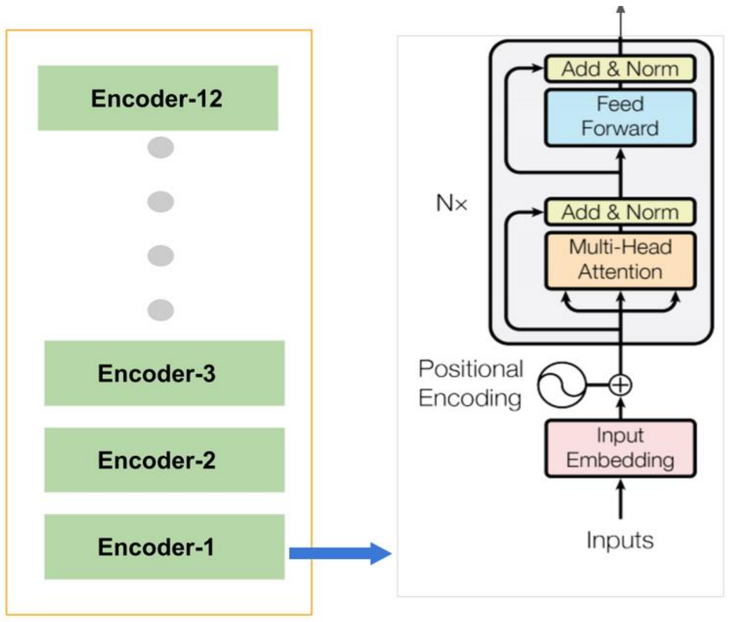


Figure 2.10: BERT architecture [44]

BERT Classifier architecture uses BERT embedding from pre-trained BERT model followed by a classification layer such as a dense layer with softmax or sigmoid activation to perform tasks. This model setup is especially effective in binary and multi-class classification tasks such as sentiment analysis, spam detection, and duplicate bug report detection.

For training the BERT classifier on text data, the first step in using BERT training is tokenization. It breaks the text into small units compatible with the model's vocabulary. BERT uses a special tokenizer known as WordPiece tokenizer. In addition to standard tokens, BERT adds special tokens to each input sequence. In this purpose special token are like [CLS] for starting of text and [SEP] is end of the text. [CLS] a classification token added at the beginning of the sequence. The final hidden state corresponding to this token is used for classification tasks. [SEP] a separator token used to distinguish between two sentences or segments in input. After tokenization, the text is converted into numerical representations using token IDs, segment IDs, and attention masks, which are then fed into the BERT model. This is particularly useful in tasks like question answering or next sentence prediction. It extracts features by itself. These features are converted into numerical form [45]. After this preprocessing data is given BERT classifier for training the model. BERT understands the pattern of the text by itself easily then just fine-tuning BERT for the task[46].

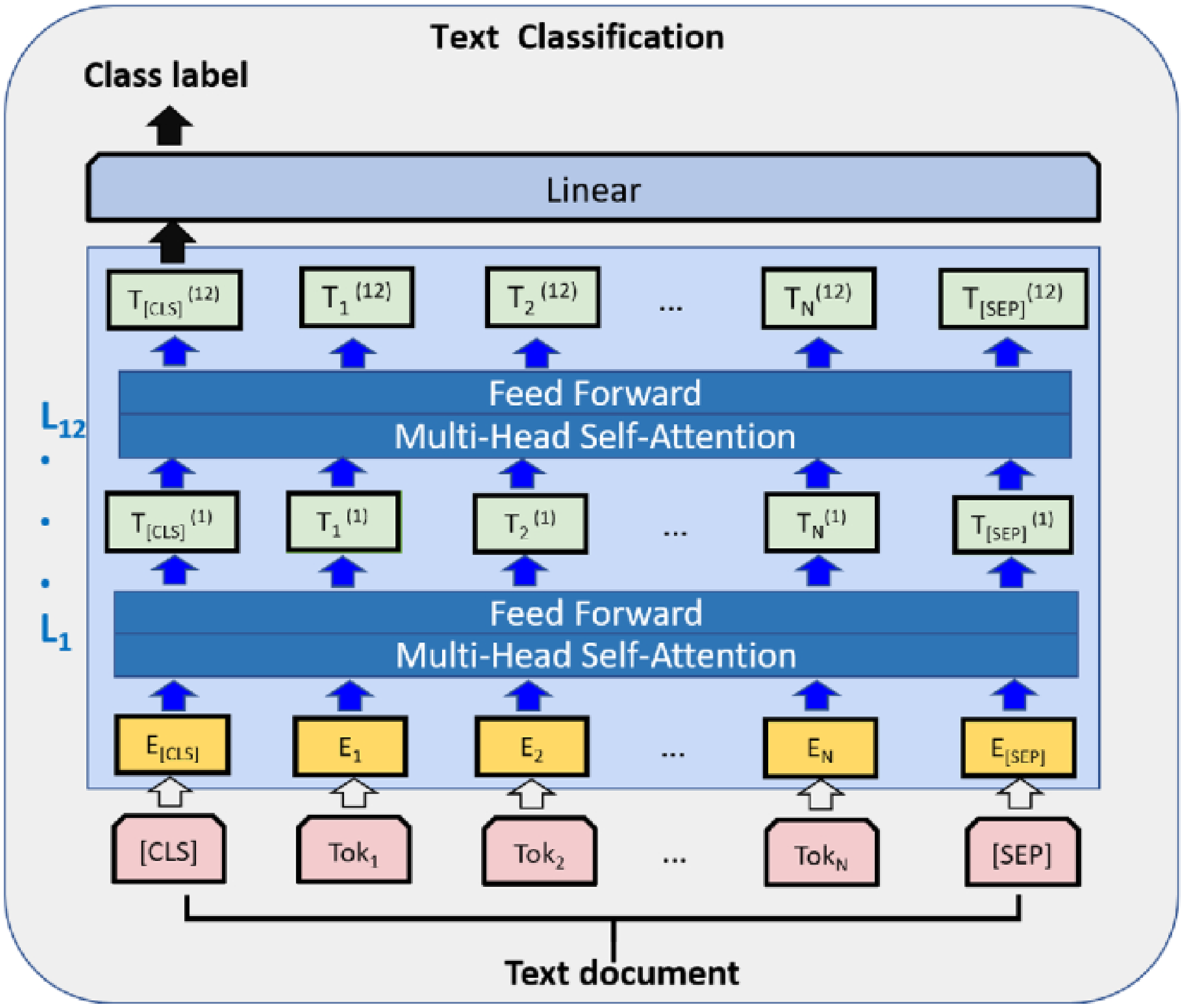


Figure 2.11: BERT architecture layers [47]

Figure 2.11 explains the internal structure of BERT's encoder layers in a clear way. Each encoder layer is made up of two main components: a multi-head self-attention mechanism and a feed-forward neural network. Each layer includes residual connections, which help the model retain important information, and layer normalization. It stabilizes the learning process. The output produced by one encoder layer is passed on as input to the next layer, enabling BERT to gradually learn more abstract and complex features from the input text data.

Since the release of the original BERT model, several improved and optimized versions have been introduced to overcome limitations like large model size and slow inference time. Bert-base-uncased (standard BERT model), Distil BERT-base-uncased (smaller, faster version of BERT), RoBERTa (improved version of BERT) and T5/Pagasus (Transformer-based models for text generation and classification).

BERT-Base-Uncased: The standard, lowercase version of BERT where all input text is converted to lowercase and stripped of accent markers.

DistilBERT: A distilled, compact version of BERT that is smaller and faster while retaining about 95% of BERT’s performance. It has 6 layers instead of 12, making it suitable for deployment in environments with limited computational resources.

RoBERTa (Robustly optimized BERT approach): It is introduced by Facebook AI. RoBERTa modifies BERT’s training procedure by removing the NSP task, using larger mini-batches, and training on more data. It consistently outperforms BERT on several benchmark datasets.

ALBERT (A Lite BERT): It is developed by Google. It is used to reduce model size by sharing parameters across layers and factorizing embedding matrices. It can achieve competitive performance.

T5 (Text-to-Text Transfer Transformer) and Pegasus: These are transformer-based models focused on text generation and summarization. T5 treats every NLP problem as a text-to-text problem and Pegasus is specifically pre-trained for abstractive text summarization using a novel pre-training objective.

2.13 Data Augmentation (DA) and techniques

Data augmentation is a technique that synthetically enlarge the size of the data set. It gives the diversity to the data set and prevent data scarcity. It is also useful to manage class imbalance. It resolve the overfitting issue in ML models [48].

When we have less number of training data set we can use data augmentation technique to enhance the size of training data set. It will help to improve the performance of ML model. There are following augmentation techniques. Easy data augmentation, Back translation.[49]

2.13.1 Easy Data Augmentation

EDA has further following operation. Random Insertion (RI), Random Deletion (RS), Synonym Replacement (SR) [50][49].

Synonym Replacement: It replaces words with their synonyms using WordNet tool. It is also called lexical substitute.

Random Insertion: Inserts synonyms of existing words at random positions. This word should not be stop word.

Random Swap or Deletion: Using this method we can randomly swap characters. We can delete any word.

2.13.2 Back Translation

Back translation means to generate data with different wording and translate it in another language. Here we can change language using translation tools like google translation API or Bing translate. It makes the changes in sentences or phrases [49].

2.14 Data Augmentation Tool

We use different tools and libraries for data augmentation. They are following. Textattack, Nlpaug, Googletrans, NLP Albumentations and TextAugment [51]

2.14.1 Text Attack

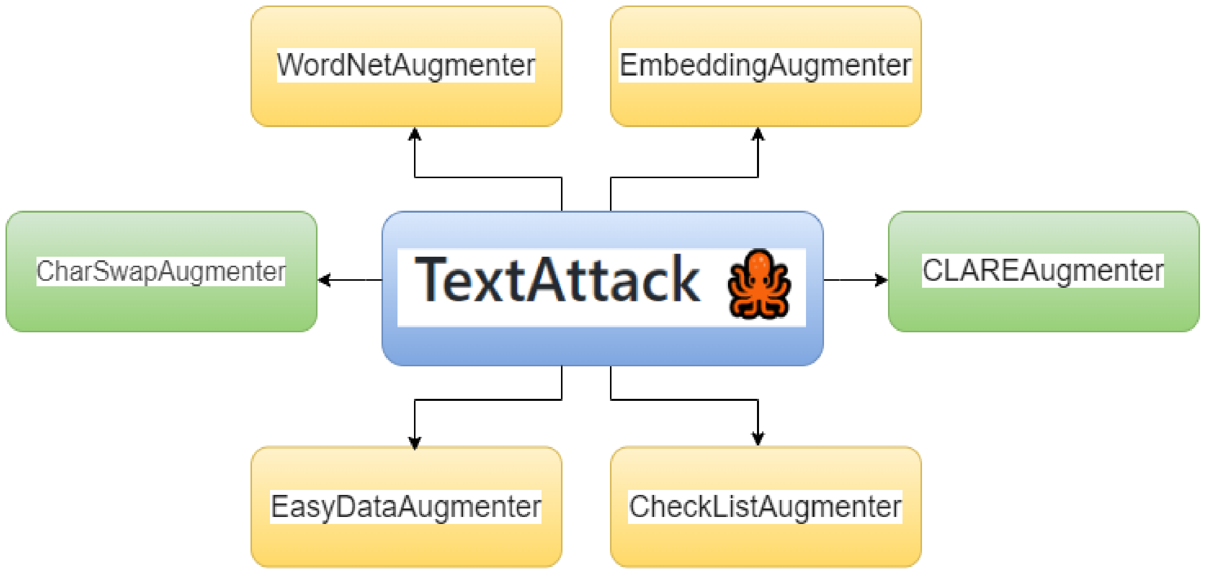


Figure 2.12: Methods of Data Augmentation [52]

Word Augmenter: It increases data by making changes in words and replacing words using WordNet. This technique leverages lexical databases to find synonyms or related words, thus enriching the dataset. By diversifying vocabulary, it helps models generalize better and reduces overfitting to specific word choices.

Embedding Augmenter: Word embedding mean change the textual data into numerical form or vectorization form. Embedding augmenter increases dataset size by selecting words with similar vector representations. Cosine similarity above 0.8 ensures that substituted words remain contextually and semantically relevant. [49].

Easy Data Augmenter: It simply insert, delete or replace the data to increase its size. This method is easy to implement and introduces random variations in the dataset. It helps simulate user errors and informal writing, which improves model robustness[53].

Char Swap Augmenter: It increases the data size by simply swapping the character. In this we simply can add or delete words.

Checklist Augmenter: It increases data by making sentences long, short and swap using different methods like name, location, number.

Clare Augmenter: It also helps in increasing data by adding new text, replacing the text with similar one. Synonym can be written for this purpose. It integrates the data with pre-trained masked language models.

2.15 Existing studies for duplicate bug report detection

In recent decades, numerous machine learning approaches have been put forward by researchers to enhance the accuracy of identifying duplicate bug reports. With the recent advancements in this field, additionally, academics have developed a number of methods for detecting duplicate bug reports that use deep learning techniques. It is commonly known that the size of the training dataset has a significant impact that deep learning-based techniques have great impact. Deep learning approaches have displayed potential when applied to bug repositories featuring a substantial volume of problem reports.

The industry makes use of various extra techniques. It's difficult to gauge how far we've come because there isn't enough area to compare them. The elements influencing DBRD performance were investigated. It was demonstrated that DBRD techniques exhibited significant differences in performance between recent and older data. It is evident that developers today would derive limited benefit from a DBRD technique excelling with data from many years ago but faltering with more recent data [5].

Automatic DBR detection methods are introduced based on several criteria for determining if the newly submitted report is a duplicate report or not. Especially machine learning techniques are used to detect the DBRs.

For detecting duplicate, cosine similarity is used but a new methodology Manhathan distance similarity approach is used for feature extraction. It helps to enhance the accuracy for detecting duplicate bug report. Decision tree classifier performed well due best feature extraction. It provided high accuracy [54].

An approach named as Detecting Duplicate bug reports with convolutional neural network has been used. The main purpose of this approach is to use semantic and syntactic features for model accuracy. Hadoop, Hdfs, Mapreduce and Spark data set have been used for it. DBR-CNN achieved accuracy 0.925, 0.954, 0.823 and 0.944 for mapreduce, hdfs, Hadoop and spark respectively [55].

Deep learning model like Convolutional Neural Network (CNN) is used for DBRD. The performance of proposed system is evaluated on datasets that are publicly available. This technique performed well and achieving accuracy rates ranging from 85% to 99% and recall@k rates between 79% and 94%. Furthermore, cross-training datasets from both the same and separate domains were used to evaluate the technique's performance. When applied to datasets from the same domain, the suggested technique produces high accuracy; however, when used to datasets from different domains, the accuracy is lower [56].

An approach Dual-Channel Convolutional Neural Networks (DC-CNN) is presented for identifying duplicate bug reports. Word2vec is used for vocabulary then single convolutional matrix is generated. CNN model is trained and predict duplicate bug reports. This approached is tested on three large data set Open Office, Eclipse, Net Beans. This model performed very well, with accuracy scores of 0.9429, 0.9685, 0.9534, and 0.9552, which were even better than other methods that use deep learning [57].

Bug reports contain structured and unstructured data. Feature extraction is best for unstructured data so a new approached feature extraction model has been used for detecting duplicate bug report. This approach used Term Frequency and Inverse Term Frequency (TF-IDF) for better feature extraction using uni-gram and bi-gram. Decision tree classifier performed well on proposed approach and improved 2% accuracy and precision and 4.5% and 5.9% recall and f-1 measurement results [58].

Another comparative analysis between deep learning techniques has been conducted. The purpose of this technique to explore different embedding models like TF-IDF, Gensim, Fasttext, BERT and ADA. According this BERT performed very well instead of other models regarding recall [59].

Another transformer based approach has been used which improved the duplicate bug report detection. This technique used unstructured information of bug report as input. BERT MLP model performed best with comparison of DC-CNN and sentence BERT. 92.11 %, 94.08%, 89.03% accuracy achieved using BERT MLP model on Mozilla, Eclipse and Thunderbird dataset respectively [60].

Another approached has been use on transformer models, it combines retrieval and classification approached for detecting duplicate bug reports. Sentence BERT and RoBERTa outperformed as compared to other models in retrieval and classification. This technique is applied on five different data sets. It performed well for time efficiency and accuracy as well [61]

A new method is introduced for identifying the duplicate bug report. It utilizes BERT model as a foundation for constructing SBERTs for both titles and content. By processing each item separately and optimizing SBERT with reports, the proposed system outperformed the baseline systems, demonstrating that it is possible to discover duplicate bug reports. [62].

The aim of study on DBRD is to lessen the workload of developers and improve the working of software bug tracking system by differentiating between duplicate and new bug reports. It employs topic modeling and a feature selection algorithm to extract relevant information, then utilizes the BERT algorithm for accurate prediction, streamlining bug report management. The proposed model achieved an accuracy of approximately 89.85% in Mozilla, 87.03% in Apache, 87.67% in Eclipse and 88.95% in KDE. These results represent a significant boost over the baseline, with improvements of roughly 44.46% in Mozilla, 47.77% in Apache, 45.17% in KDE and 36.33% in Eclipse [6].

Maintaining industrial software is vital but complex. Identifying duplicate bug reports, typically done manually, can be enhanced by employing automated methods. This approach involves vectorising report content and using deep learning-based sentence embedding for similarity assessment. The system's performance, fine-tuned with sentence-BERT, is empirically compared to benchmarks, demonstrating its superior effectiveness in locating duplicate bug reports [63].

A novel approach for detecting Duplicate Bug Reports (DBRs) is introduced, known as the CTEDB method. This method involves selecting key information from Eclipse and Mozilla projects. The subsequent step involves assessing the similarity of the technical terms extracted from the bug report text data. The similarity calculation findings are coupled with the binary classification confidence of the DeBERTaV3 model to provide the final score that is used to evaluate the bug report duplication once the technical terms have been extracted. The experiment compares the use of a dual-channel convolutional neural network and a conventional convolutional neural network to identify Duplicate Bug Reports (DBRs) in order to evaluate the accuracy of the outcomes [[64].

Different embedding models are examined and compared. How can accurately identify the duplicate bug report on the base of textual similarities. Such as TF-IDF (Baseline), Gensim, BERT, FastText and ADA. In general, BERT outperformed the other models, especially in terms of recall [65].

This study aimed to enhance the integration of automated duplicate bug report detection techniques into a tester's workflow. They proposed Bugle as an automated tool to present relevant bug reports. They achieved an average success rate of 94.44% in locating duplicates based on participants' queries. Participants achieved a 75.00% accuracy in identifying duplicates from the reports Bugle retrieved [66].

Duplication in bug reports poses a significant challenge in large open-source projects. To overcome this challenge they introduced a novel machine learning based approach for automatically detecting duplicate bug reports using textual data. They explored six distinct methods: Topic modeling, Gaussian Naïve Bayes, deep learning, time based organization, clustering, and summarization via a generative pre-trained transformer large language model. They introduced a new method for finding duplicate bug reports. This method uses a threshold-based approach instead of the usual "top-k" selection. They tested their methods on a public dataset from an Eclipse open-source project. It showed good accuracy ranging from the high 70% to 90%. The multi-layer perceptron neural network model performed the best [67].

Duplicate bug report detection is difficult when the reports are written in different ways or are very short. According to existing studies 23% of duplicate reports don’t look similar in word. They just mean the same thing which traditional methods struggle to identify.78% of bug reports in open-source projects are short and use technical terms that make the task even harder. They used over 92,000 bug reports from three open-source systems and tested seven existing detection techniques to tackle these problems. They found that when reports were enriched, these techniques performed much better [68].

A novel technique CorNER has been introduce. It has primary goal of significantly improving the accuracy of duplicate bug report (DBR) detection. This technique tackle the challenge of unstructured bug report information by converting the into structured data format. This technique initially employs Random forest with context (RNER). This RNER component is specifically designed to identify and label important entities of bug report like titles and their detailed descriptions. This technique used Text Convolutional Neural Networks (TextCNN) for feature extraction after using Random Forest with context (RNER) [69]. CorNER is able to process bug reports more effectively. It ultimately achieves much higher precision in identifying duplicate bug reports.

Deep learning-based methods have performed well in bug repositories with a high volume of bug reports. However, the available deep learning algorithms perform worse than the traditional approaches in bug repositories with a normal number of issues [5][70].

Data augmentation has different techniques for natural language processing. Easy data augmentation technique is one of the best techniques. It reduces the overfitting and boost the performance of model on text data [50]

A useful method for dealing with the problem of small sample sizes in many natural language processing (NLP) jobs is text data augmentation. It helps to enhance the performance in form of accuracy of machine learning techniques [71]. Text classification performance is significantly improved by data augmentation techniques. It incorporates four key operations: random deletion, synonym replacement, random insertion and random swap [72]. It helps to increase the bug reports.

This study centers on spotting duplicate bug reports in projects with small bug report numbers. A novel approach named "Cupid" combines traditional methods with advanced language models like ChatGPT. Cupid uses ChatGPT to identify crucial keywords and pairs them with the REP method. Across three datasets, Cupid outperforms other techniques, achieving scores between 0.59 and 0.67 in Recall Rate@10 [73].

2.16 Summary

This chapter gives a detailed review of the research background related to the study, including a comprehensive examination of existing literature on bug reports, duplicate bug report detection, data augmentation techniques, machine and deep learning models. It explores various methods and approaches previously proposed by researchers to identify and manage duplicate bug reports effectively. The chapter discusses the challenges commonly faced in this area, such as data imbalance, semantic similarity, and limited labeled datasets. It examines different data augmentation strategies that have been used to enhance training data and improve model performance. By analyzing past studies and comparing their outcomes, this chapter establishes the research gap and demonstrates the need for an improved and more effective solution.

CHAPTER 3

RESEARCH METHODOLOGY

This chapter lays out the research methodology for finding duplicate bug report detection. It is helpful in detecting duplicate bug reports when we have small project or small number of data set of bug reports. Basically, in this research, an experiment has been conducted to address the research question effectively. This experiment follows a well-defined sequence of steps like data acquisition, preprocessing, data augmentation, model selection, implementation and evaluation of model. To address the research problem, the following methodology is proposed. Figure 3.1 explains the research methodology and experimental process.

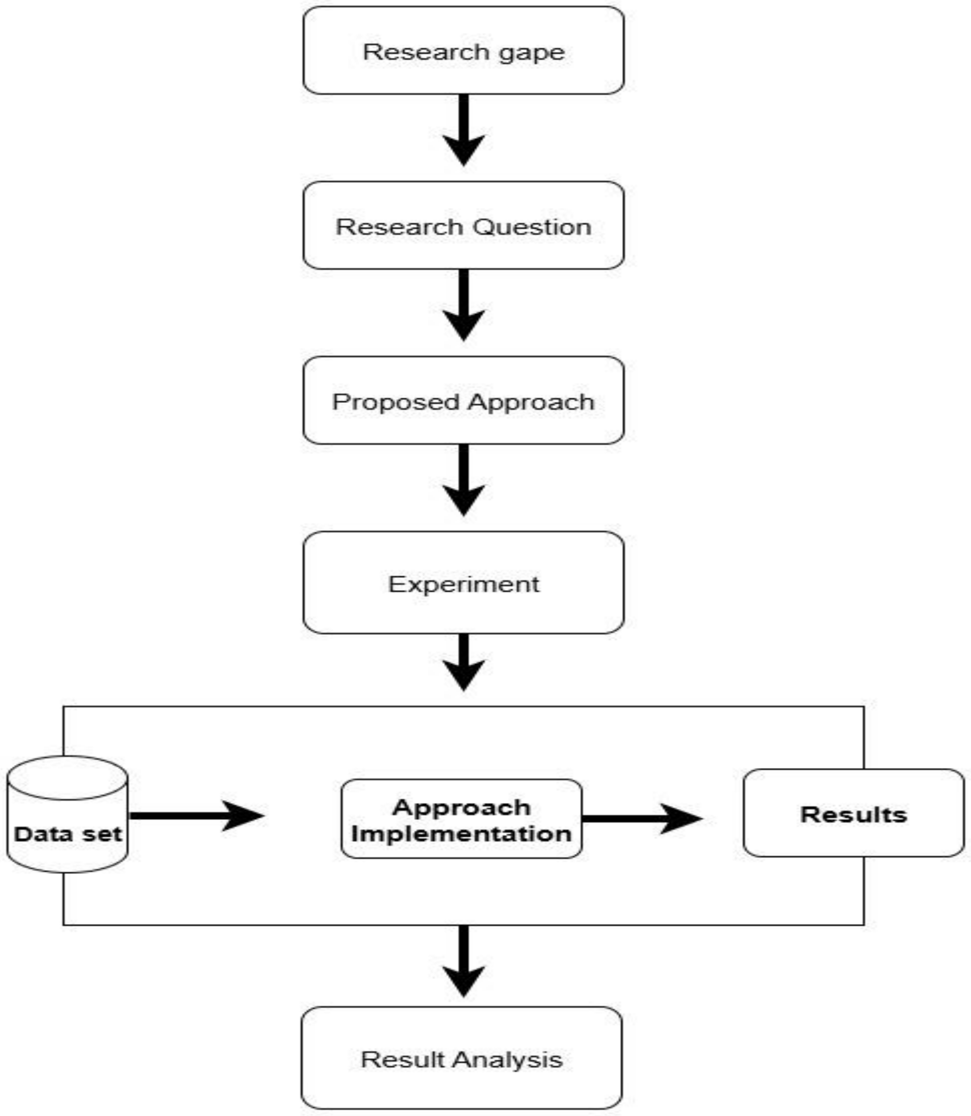


Figure 3.1: Research Methodology and Experimental Procedure.

3.1 Data Collection

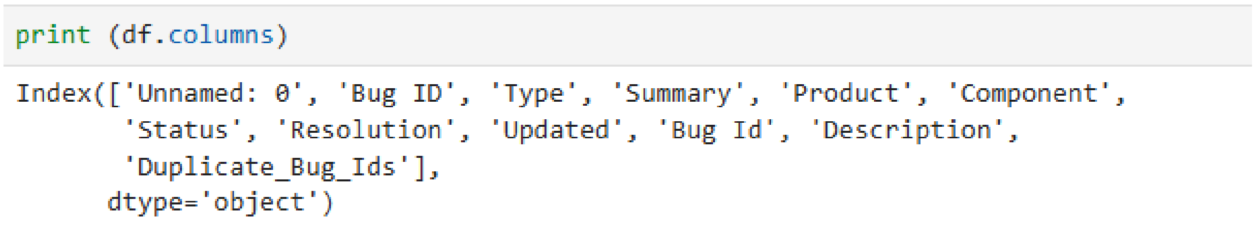
The first step in conducting the experiment is data acquisition, specifically bug reports. Bug report data is the requirement. Bug report data has been collected from various open-source software systems, including Eclipse, Firefox, and Focus for iOS and Gecko view for Android version. Data is stored on popular bug tracking system called Bugzilla [74]. This data set has been used in different existing studies [75] [76][77] [78]. This particular dataset featuring bug reports exclusively from mobile applications. All available bug reports from Firefox for iPhone, Focus for iPhone, and GeckoView for Android have been compiled and gave it the name "Mobile Dataset". The main reason for choosing this specific dataset for experiment is its limited number of bug reports. Table 3.1 shows detail about data set.

Table3.1: Data set [74]

Bug Reports Dataset (2017 – 2022)	**Number**
Whole Dataset (Bug Reports)	5320
Duplicate Dataset (Bug Reports)	562
Non-duplicate	4758
Duplicate Ratio of Bug	10.56%

3.2 Targeted Features from dataset

Bug report is an important document that contains following information about Bug like Bug ID, Type, Summary, Product, Component, Status, Resolution, Description and Duplicate Bug ID. For the research experiment, following features have been selected from the dataset that are type, summary and description. Type is a categorical and structured data type. It classifies the nature of bug. Summary and description are textual data and unstructured data type. On the basis of these features we can predict that bug report is duplicate or not. So the target column is Duplicate_Bug_ids.



```
print (df.columns)
```

```
Index(['Unnamed: 0', 'Bug ID', 'Type', 'Summary', 'Product', 'Component',
       'Status', 'Resolution', 'Updated', 'Bug Id', 'Description',
       'Duplicate_Bug_Ids'],
      dtype='object')
```

Figure 3.2: Data set columns

3.3 Data Preprocessing

Before training data set, data preprocessing is an essential part of NLP. There is requirement of preprocessing in natural language processing. It organizes the data and makes it more meaningful and noise free. Following steps have been followed for data preprocessing that has been explained in chapter 2.

3.3.1 Loading Dataset

Information was extracted from CSV data file using Pandas library.

3.3.2 Data Cleaning

In data clearing, firstly all alphanumeric characters like punctuations, html tags and stop words were removed to make data set more meaningful and understandable. Missing values were handled by either dropping or filling the values. Irrelevant data entries were discarded and the duplicates values were removed. Finally the text was in standard form [14].

3.3.3 Tokenization

The summary and description fields in bug reports were in textual form, so tokenization was applied to process them. Tokenization split the text into individual words or tokens, making it easier for machine learning models to understand and analyze [29]. Tools such as NLTK, spaCy, and Hugging Face tokenizers were used for this purpose. For CNN and LSTM tokenization is performed on word level and for BERT use special token like ([CLS], [SEP])

3.3.4 Feature Engineering

Feature engineering was a crucial step in the Natural Language Processing (NLP) workflow, as it transformed raw text data into a numerical vectored format suitable for machine learning and deep learning models. In this study, three deep learning models have been selected. CNN, LSTM, and BERT were utilized. For CNN and LSTM models, the TF-IDF vectorizer was employed for feature engineering. It is used to convert textual data into numerical features[29]. For BERT model, Pre-trained BERT embedding were used to encode the semantic and contextual meaning of the text. BERT embedding are more accurate numerical representation and it improves the model’s performance.

3.4 Data Augmentation

In order to address the challenge of limited bug number of data set, data augmentation technique was applied. Augmentation technique artificially increases the data size that is helpful in the training model. Many augmentation techniques were applied to synthetically expand the dataset without making any original semantic meaning of the text [79]. To accomplish the whole augmentation process, the TextAugmenter class was introduced. This class specifically designed to handle text augmentation and tackle class imbalance in text classification tasks. It can be applied particularly for duplicate or non-duplicate classification problems.

The TextAugmenter class was set up using a pandas data frame. This data frame included both the text data and their labels. The class takes in:

A Data Frame (df) that holds the data

The name of the text column (text_col, default is 'text_data')

The name of the label column (label_col, default is 'is_duplicate')

When the class was first initialized, it checked and showed the original distribution of class labels. This gave us a clear picture of the data before any changes were made. It also helped us spot any imbalance between duplicate and non-duplicate reports.

When the augmentation process began. It created new, diverse and synthetic samples by using NLP based techniques. These included replacing words with synonyms, randomly inserting new words, and shuffling sentences [80] [81].

We used the following augmentation methods to increase the size of our dataset [82].

3.4.1 Synonym Replacement

The synonym replacement method was employed as a data augmentation technique to enhance the diversity and size of the textual dataset. This approach involved replacing a predefined number of words (n) in each sentence with their corresponding synonyms [49]. We tried to make sure the meaning of sentence stayed same. This technique proved particularly useful in tasks such as duplicate bug report detection, where the same idea may be conveyed using different vocabulary. The synonym replacement process followed a structured procedure:

In the first step, the input text was tokenized into individual words or tokens. We randomly chose some selection of words for synonym replacement from the tokenized sentence. Preference was given to longer words because it carry more semantic weight than shorter or functional words. The selected words were then passed through WordNet. It is a widely used lexical database. We can retrieve appropriate synonyms. In the final step, the original words were replaced by their synonyms. These changes gave a new version of the sentence with slightly altered wording but retained intent [71].

To ensure consistency and control over the augmentation process, the following fixed parameters were applied:

A default of n = 2 replacements was performed for sentences classified under class 0 (non-duplicate).

A slightly higher n = 3 replacements were carried out for class 1 (duplicate) sentences to introduce more variation, as duplicates are often expressed in different phrasings.

Only words longer than 3 characters were considered for synonym substitution to avoid replacing articles, conjunctions, or common short words that could negatively affect sentence structure.

Only alphanumeric words were included in the replacement process, excluding punctuation marks, numeric values, or special characters.

3.4.2 Component Shuffling

The method worked by shuffling components of a sentence that were separated by specific delimiters. It was particularly effective for texts with naturally segmented structures, such as descriptive fields or formal documentation. The following patterns and strategies were employed:

It handled colon separated segments, commonly found in structured entries (e.g., "Title: Description: Additional Info"), by reordering these segments to generate new sentence forms.

It also shuffled components of the sentence that were separated by common prepositions such as in, at, with, by, for, and on. This allowed variation in the placement of phrases without altering the core meaning.

The method ensured that the main content or key information of the sentence remained intact while its structure was modified, thus supporting semantic preservation.

To guide the augmentation process and maintain linguistic coherence, the following fixed values were applied:

A regular expression pattern was used to identify and split text based on the presence of relevant prepositions. This ensured precise segmentation for shuffling.

If no suitable splits were found (i.e., the sentence did not contain colon-separated parts or prepositions matching the pattern), the original sentence structure was preserved to avoid unnecessary distortion or semantic drift.

3.4.3 Random Insertion

The random insertion method inserts n words that are synonyms of existing words: Finds candidate words from the original text, Generates synonyms using WordNet and Inserts synonyms at random positions

To ensure control and consistency, the following fixed values were applied during this augmentation technique:

A default of n = 2 synonym insertions was used for class 0 (non-duplicate) sentences.

A more aggressive augmentation of n = 3 insertions was applied to class 1 (duplicate) sentences, as these benefit from greater lexical variety due to their semantic overlap.

Only words longer than 3 characters were considered valid candidates for synonym generation to avoid inserting function words or short, context-independent terms.

Additionally, only alphanumeric words were eligible, ensuring that symbols, numbers, and punctuation marks were excluded from the augmentation process.

3.4.4 Class Balancing Strategy

Class balance strategy is very important when we have unevenly distributed data set. One class is minority or second one is majority class. Model performance get improve for majority because of more training data set. This situation can cause poor classification [48]. The balance class method generates synthetic samples to reach a target count for each class. The method calculates how many samples need to be generated for each class based on the difference between the current count and the target count.

We have total 5320 number of bug reports. 4758 are non-duplicate and 562 are duplicate bug reports. In our data set we have less amount of duplicate bug report as compared to non-duplicate bug reports. So predicting duplicate bug report is comparatively difficult because of less amount of training data. To solve this problem we have following key parameters.

Key Parameters: Fixed Value for Target Count, Default target_count=10000 samples per class and this parameter can be adjusted based on requirements.

3.4.5 Augmentation Rotation Strategy

We have two classes. Duplicates and non-duplicates.

For class non-duplicates we say it 0. We uses 4 different augmentation techniques in rotation. Here (n) represents number of samples:

1. Synonym replacement (n=2)

2. Component shuffling

3. Random insertion (n=2)

4. Combined approach (synonym replacement + shuffling)

For class duplicates we say it 1. We use five different augmentation techniques in rotation (more aggressive):

1. Synonym replacement (n=3)

2. Component shuffling

3. Random insertion (n=3)

4. Double synonym replacement

5. Multi-combined approach (synonym + shuffle + insertion)

3.4.6 Main Augmentation Interface

The augment method provides the main interface for users. This method accepted the following two key parameters:

Balance (default=true): This Boolean parameter controlled whether the dataset should be balanced across classes after augmentation. When set to True, the method ensured that each class in the dataset contained an equal number of samples, which is particularly useful in scenarios with class imbalance, such as binary classification tasks.

target_count (default=10,000): This integer parameter defined the desired number of samples per class. It specified how many total augmented instances should exist for each class after augmentation. This value was passed to internal methods to guide the extent of augmentation.

If balance is set to True, the method calls balance_classes with the specified target count. Otherwise, it adds a fixed number of augmentations for each original sample without attempting to balance the classes.

3.4.7 Data Generation Process

The data generation process was a structured and iterative approach designed to balance the dataset and enhance its overall size through systematic application of augmentation techniques. This process ensured that both duplicate and non-duplicate classes were well represented, addressing the issue of class imbalance which can adversely affect the performance of machine learning models. The following key steps were followed**:**

Initial Analysis:

The process began with an analysis of the **initial distribution** of the dataset classes. The distribution revealed a significant class imbalance:

Class 0 (non-duplicate bug reports) consisted of 4,759 samples.

Class 1 (duplicate bug reports) consisted of only 562 samples.

This imbalance highlighted the need for a targeted augmentation strategy to generate synthetic data for both classes in order to achieve uniform representation.

Generation Planning:

After analyzing the class distribution, the system calculated the number of synthetic samples required to meet the predefined target of 10,000 samples per class. The augmentation goals were set as follows:

For **Class 0**, with 4,759 original samples, an additional **5,241** synthetic samples were required.

For **Class 1**, with only 562 original samples, a much larger number of **9,438** synthetic samples had to be generated.

This planning stage was crucial to ensure a balanced and comprehensive dataset suitable for model training and evaluation.

Sample Generation:

To generate the synthetic samples, the process followed a randomized sampling approach:

For each required augmentation, an original text sample was selected at random with replacement from the existing dataset.

One or more augmentation techniques were applied based on a rotational strategy, cycling through different methods (e.g., synonym replacement, shuffle components, random insertion, character swaps) [80].

The newly generated text was stored along with associated metadata, maintaining traceability and allowing further filtering or analysis if needed.

This ensured diversity in the augmented data while retaining the core semantics of the original bug reports.

Metadata Tracking:

Each generated sample was meticulously tracked using metadata fields to distinguish between original and synthetic data. The metadata included:

The original class label (0 or 1).

An augmentation status flag, set to True for synthetic samples and False for original ones.

The type of augmentation applied, such as "synonym", "shuffle", "insertion", "character swap", or "combined".

Verification:

After the data generation process was completed, a final verification step was performed. This included:

Confirming that the target sample count for each class (10,000) was met.

Reporting the final distribution across both classes to ensure balance.

Reviewing the total sample size and ensuring all augmented data was properly labeled and stored.

This comprehensive verification validated the effectiveness and correctness of the augmentation process and prepared the dataset for downstream machine learning applications.

3.5 Model Selection

Based on insights from the literature review and the nature of the problem, three deep learning models have been opted for this study: Convolutional Neural Network (CNN) [83], Long Short-Term Memory (LSTM), and Bidirectional Encoder Representations from Transformers (BERT) [80]. These models are widely recognized for their strong performance in various NLP tasks, especially text classification and semantic similarity. We applied these models to both the original (non-augmented) and the synthetically expanded (augmented) datasets. The goal was to analyze their performance in detecting duplicate bug reports. The figure 3.3 illustrates the architecture of the BERT model used in our experiment. The input bug report text undergoes preprocessing, including tokenization, stop-word removal, and stemming. These inputs are then embedded and passed into the BERT encoder, which produces contextualized embedding. These are followed by a SoftMax classification layer that outputs whether the bug report is duplicate or non-duplicate. Here is working of BERT model [84], which we applied for duplicate bug reports detection.

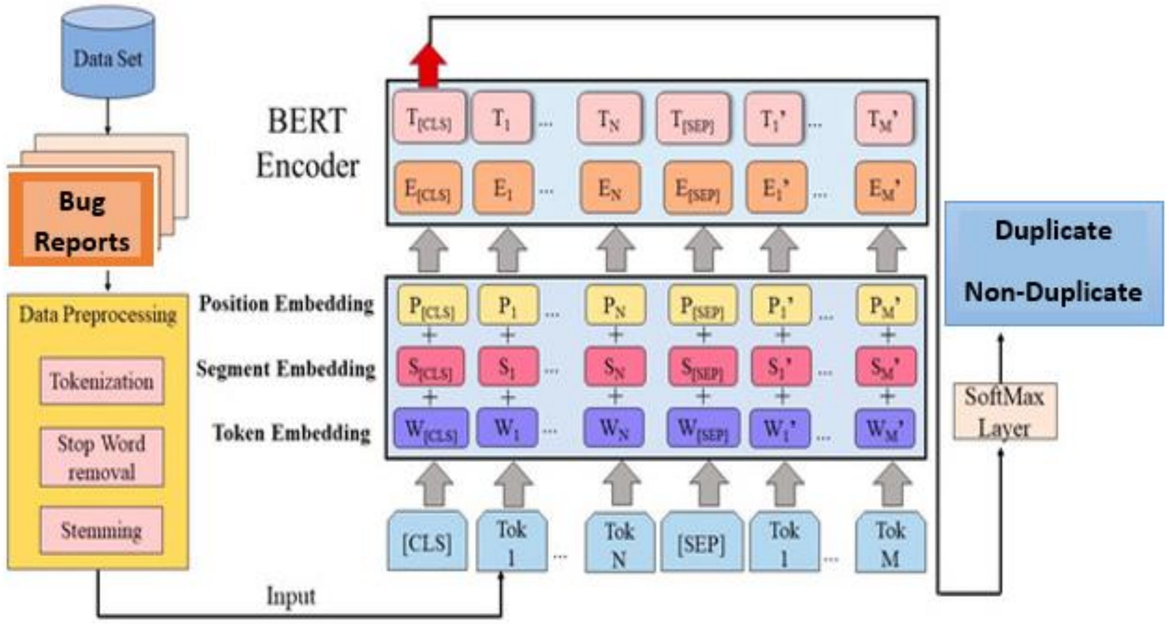


Figure 3.3: Working of BERT Classifier

3.6 Training and Testing

The dataset was systematically divided into three subsets to ensure fair and unbiased training and evaluation. 70% of the data was used for training the model, 20% for testing and 10% for validation purposes.

This split helps in both optimizing the model and validating it on unseen data. To train the deep learning models, we used the Adam Optimizer. It is an efficient and adaptive gradient descent algorithm. Fine-tuning was performed particularly on the BERT model, where the pre-trained weights were adjusted on our domain-specific bug report data. Hyper parameters such as learning rate, batch size, and number of epochs were tuned to maximize classification accuracy. Early stopping and dropout techniques were also used to prevent overfitting and improve generalization. Special care was taken to ensure the data split preserved the balance between duplicate and non-duplicate classes.

3.7 Model Evaluation

Following metrics were used for model evaluation and comparison. Accuracy, precision, recall, f1-score ROC-AUC Score and confusion matrix. Details of these model evaluation metrics have been discussed in chapter 2. This comparison is necessary to identify the best deep learning model.

3.8 Comparative analysis

After training and evaluating the CNN, LSTM, and BERT models on both the augmented and non-augmented datasets, we conducted a comparative analysis as part of our research methodology. To enhance the effectiveness of proposed approach, result are compared with existing latest studies. This step aimed to systematically examine how the inclusion of synthetically generated data influenced the learning process of each model. The purpose of comparative analysis was not only to compare machine learning model's performance or outcomes but also to understand the methodological impact of data augmentation. How it gives impact on model training and feature representation. We also studied how it influences the model's ability to generalize.

This comparison was done carefully to make it fair. We used the same preprocessing steps for all models. We split data 70, 20 and 10 ratio for training, testing, and validation. We also used the same hyper parameters wherever possible.

To reduce the effect of randomness, we ran the models multiple times. This helped us get stable and reliable performance results. We also watched how the models learned during training. We checked how the loss values changed over time. This helped us spot any signs of overfitting or under fitting in both datasets augmented and non-augmented.

We kept the experiment setup controlled to make sure the results were reliable. This helped us check if data augmentation truly improved the model in a meaningful way. It also showed whether the new data added any bias or caused the model to behave differently during training.

These findings are important. They helped in support the idea that data augmentation is a useful step, especially when working with limited data. This is particularly helpful for tasks like text classification and duplicate bug report detection [17].

3.9 Summary

Research methodology is explained in this chapter. It explained the flow of working and structured approach we followed to keep the research systematic. Experiment is designed to solve the problem of detecting duplicate bug reports. We covered the detailed process of NLP like data collection, data extraction and cleaning, data preprocessing, training and testing. Detail process of data augmentation technique is explained that we applied for increasing the data size. These steps are important to make sure the models got best preprocessed data.

The main goal of this chapter is to describe the whole experiment process that we used. It also shows that how we covered all research objectives. It gives a step by step explanation of the full process, from preparing the data to evaluating the results.

CHAPTER 4

RESULT AND DISCUSSION

This chapter covered the effectiveness of data augmentation techniques that is used to increase the number of bug report in bug tracking system. It also explained the effectiveness of data augmentation techniques for deep learning models. It systematically explores that how the augmentation process contributes to enhancing model performance.

Comprehensive analysis of the results has been addressed in this chapter that we obtained from applying deep learning and machine learning models to both augmented and non-augmented datasets.

Three deep learning models CNN, LSTM, and BERT is analyzed and evaluated comparatively in this chapter. Both augmented and non-augmented datasets have been used to train and test these models. The effectiveness of these techniques has been assessed through comparative analysis.

4.1 Overview

Machine learning and deep learning models' evaluation is an important step in the research process. It can be used to understand the extent to which models can perfectly provide insights on new data. How the model has learned from the training data can be evaluated through model evaluation. It also evaluates how the predictions can be generalized to new inputs. In order to assess the reliability, robustness and applicability of the models, this step is quite crucial. We performed the evaluation on both data set non-augmented (original, small dataset) or augmented.

For training, testing, and validation, the dataset was divided into three subsets 70, 20 and 10%. The data split is necessary to make sure that the models are trained for specific portions of the data. However, it is still analyzed on unseen examples to examine generalization performance.

4.2 Deep learning model results on Non-Augmented Data

Three deep learning models CNN, LSTM and BERT's performance on the non-augmented data through validation process is summarized below. Table 4.1 explains the result of machine learning models of non-augmented data. Five metrics of evolutions are discussed that is accuracy, precision, recall, F1 score and AUC score.

Table 4.1: Result of non-augmented data

Model	Accuracy	Precision	Recall	F1 Score	AUC Score
LSTM	88.47%	0.9039	0.9308	0.9172	0.97
CNN	89.35%	0.4000	0.0732	0.1237	0.69
BERT	89.72%	0.5000	0.0366	0.0682	0.68

4.2.1 LSTM Model

In the LSTM model, Different parameters are selected to evaluate the performance for the task of duplicate bug report detection. LSTM model can learn the complex pattern in textual data using 480 hidden dimensions. This network architecture model consisted on 3 layers. These layers are enable to capture long range dependencies in sequences effectively. 0.25 dropout rate was applied to reduce the risk of overfitting and improve generalization in model. 0.0007learning rate was setup which helped to ensure smooth and stable model learning. 64 batch size is selected to balance training speed and memory efficiency. Adam optimizer is used for training model. It is known for its adaptive learning capabilities. The model is trained for over 15 epochs to allow sufficient learning while avoiding excessive training time. The entire process helped in achieving high recall and F1-scores on the non-augmented dataset. Figure 4.1 is showing the results of training and validation process how it works.

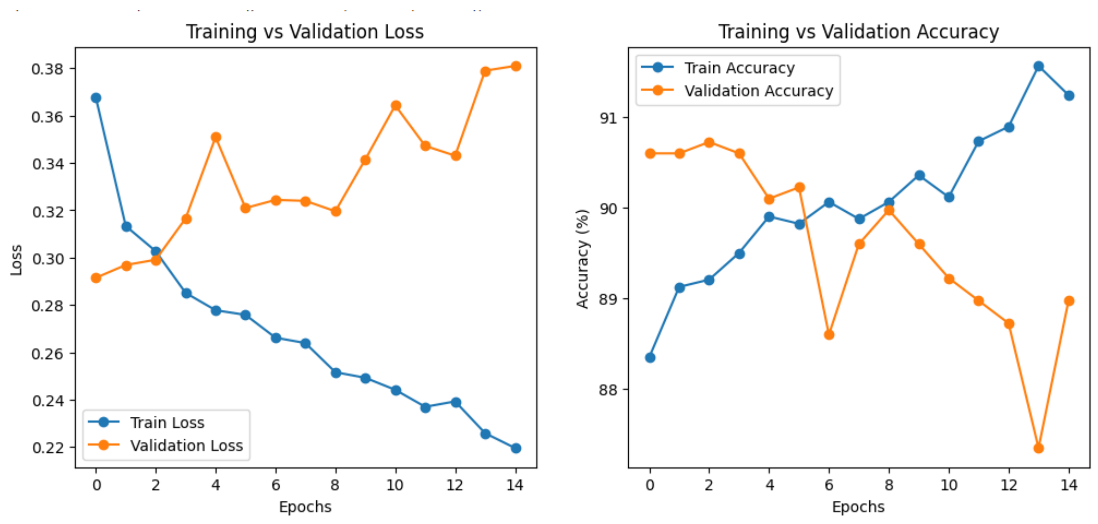


Figure 4.1: LSTM Accuracy and Loss

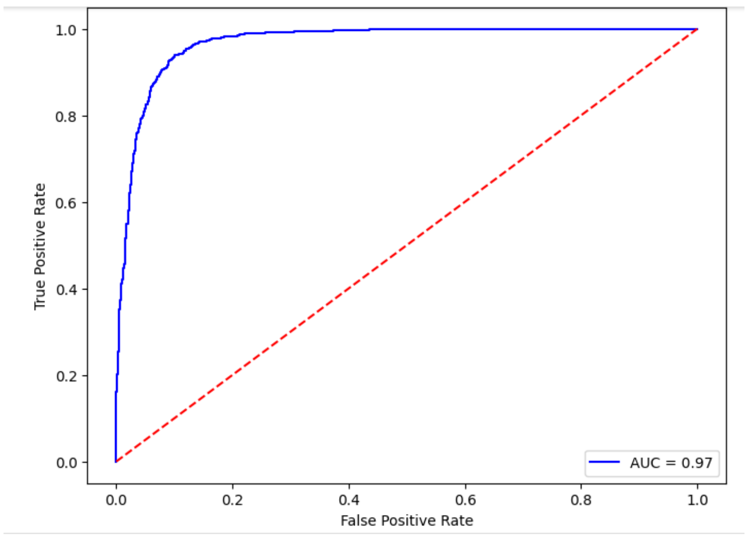


Figure 4.2: LSTM ROC Curve

Figure 4.2 shows the model performance. It showed good score**.** It means model performed well.

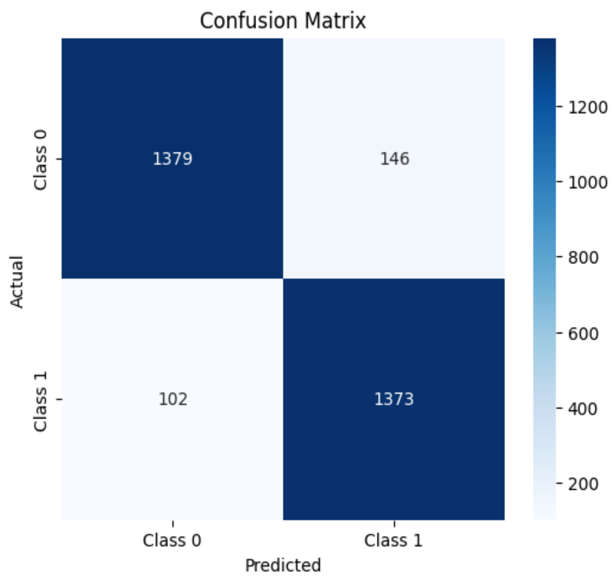


Figure 4.3: LSTM Confusion Matrix

Figure 4.3 confusion matrix shows the correct prediction of the result. LSTM predict maximum correct duplicate repots.

4.2.2 CNN Model

In the CNN (Convolutional Neural Network) model, various parameters are fine tuned to ensure best performance in detecting duplicate bug reports. 128 filters of CNN are used to capture and understand the text pattern. The model used kernel size of 3. It allowed the model to extract meaningful n-gram patterns from the input data. To extract the textual features at different point, CNN architecture used 3 layers. 0.4 dropout rate of was applied to prevent overfitting using convolutional and fully connected layers.32 batch size is selected and 0.0005 is a learning rate. It is used to keep a balance between model effectiveness and resource usage.

CNN model has fully connected layers that come after convolutional layers. It helps in reducing dimensions. It helps in compressing features before final classification. For best optimization, the Adam optimizer is employed. It helps to improve learning stability and reduce overfitting. This configuration allowed the CNN model to learn efficiently from limited non-augmented data. The model's performance is affected by the sparsity of duplicate samples. Figure 4.4 explained the training and validation loss. Figure 4.5 showed training vs validation accuracy. This pattern showed overfitting so need data augmentation.

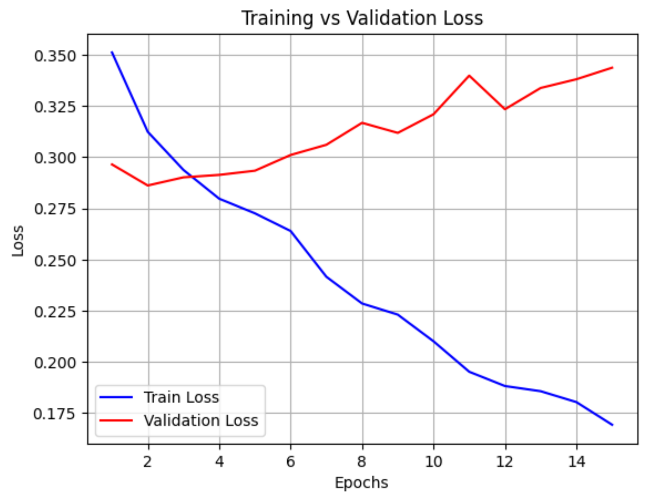


Figure 4.4: CNN Training vs Validation Loss

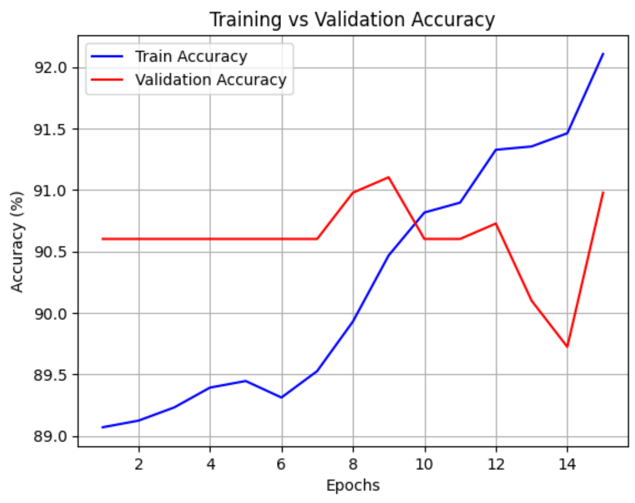


Figure 4.5: CNN Training vs Validation Accuracy

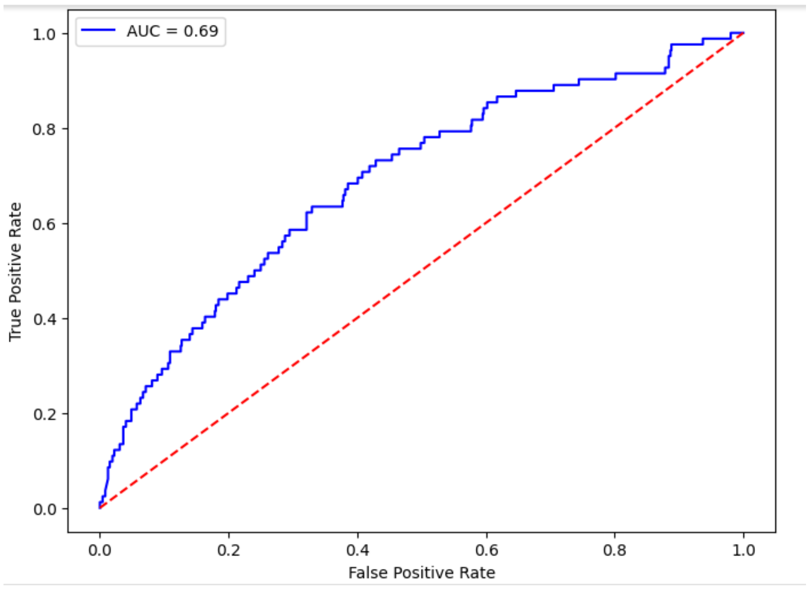


Figure 4.6: CNN ROC Curve

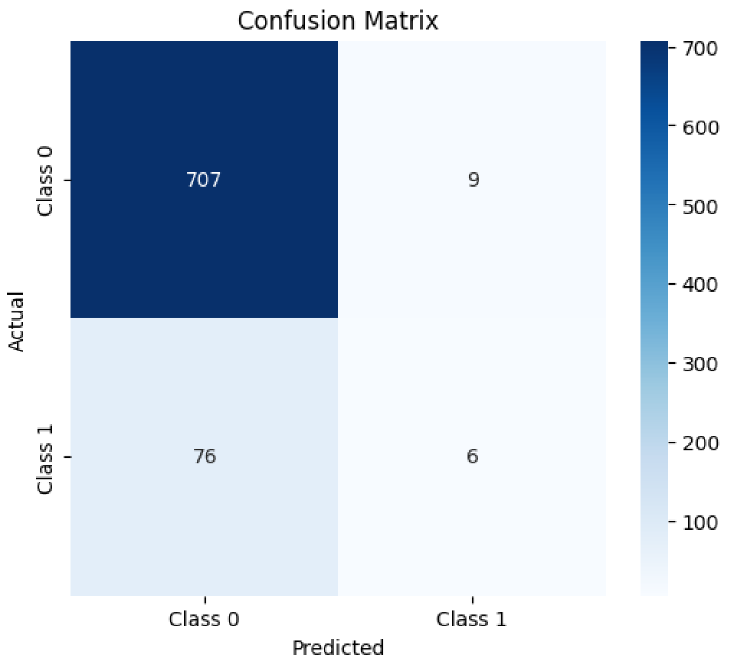


Figure 4.7: CNN Confusion Matrix

Figure 6.7 shows that model performed well on class 0 but struggled in class 1. It indicates class imbalance. To overcome this challenge data augmentation has been done to create balance class.

4.2.3 BERT Classifier

Pre-trained 'BERT-base-uncased' model is used. It is suitable for natural language understanding tasks due to its ability to capture deep semantic relationships from textual data. The learning rate is set to 2e-5. This fine-tuning transformer based model is best for training. A batch size of 8 is selected that helped in considering the high memory requirements of BERT.

A dropout rate of 0.3 is applied during training. It is useful to reduce the risk of overfitting. It improves model generalization. The model is optimized using the AdamW optimizer. It is specifically designed for weight decay regularization and has shown strong performance with transformer architectures. The hidden layer architecture was configured as 768, 256, and 2. It means the output from the BERT encoder is 768 dimensional. It went through a dense layer of 256 units before mapping to the final binary output layer. This fine-tuning approach enabled BERT to adapt to the duplicate bug report detection task effectively although it is relatively small datasets. Figure 4.8 shows the model's achievement.

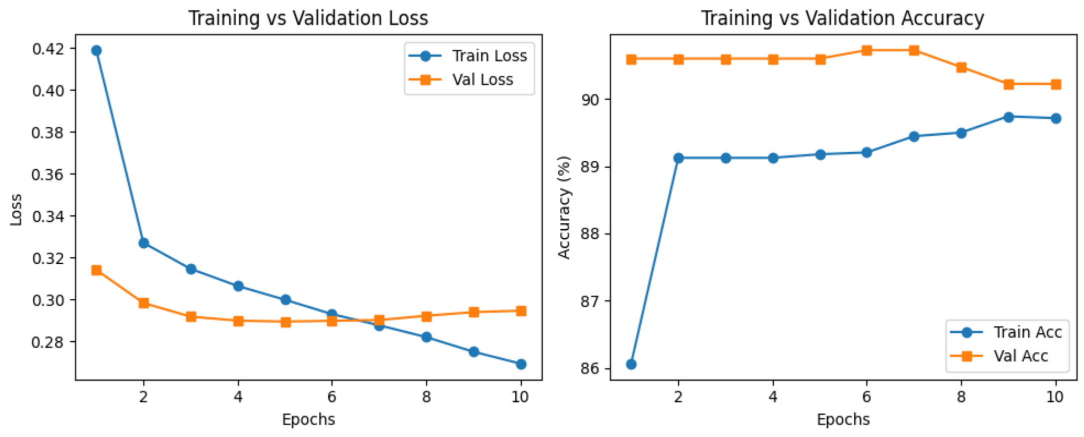


Figure 4.8: BERT Training vs Validation Loss and Accuracy

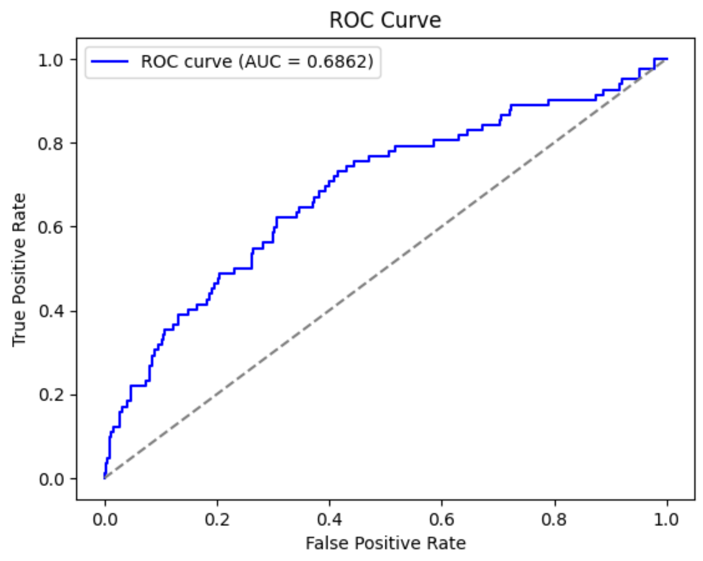


Figure 4.9: BERT ROC Curve

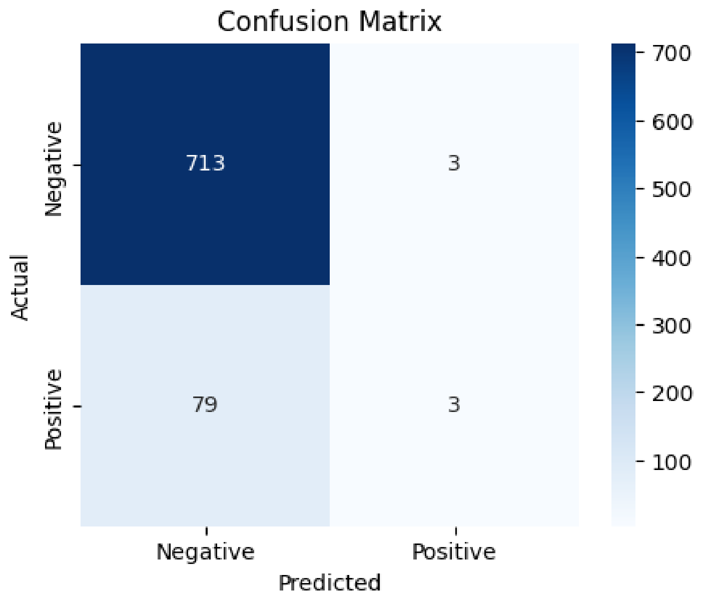


Figure 4.10: BERT Confusion Matrix

Figure 4.10 explains true negatives predicted well and false positive predicted incorrect. False negative predicted incorrect or true positive correctly predicted.

4.3 Deep learning model results on Augmented Data

There are following results of deep learning models on augmented data.

Table 4.2: Results of augmented data

Model	Accuracy	Precision	Recall	F1 Score	AUC Score
LSTM	94.77%	0.9190	0.9695	0.9436	0.98
CNN	94.77%	0.9083	0.9939	0.9492	0.99
BERT	96.30%	0.9528	0.9715	0.9621	0.99

4.3.1 LSTM Model

To evaluate the impact of data augmentation on the performance of deep learning models, we trained the LSTM model using the synthetically expanded dataset. Several hyper parameters were adjusted to enhance the model's learning capabilities. The LSTM model was configured with 64 hidden dimensions, which helped in learning relevant sequential patterns from the augmented textual data. The model architecture included 2 LSTM layers, which provided sufficient depth to capture long-term dependencies within the bug report descriptions.

Using data augmentation, data set is increased. To avoid overfitting a relatively high dropout rate of 0.6 was applied. 0.001 Learning rate is selected for efficient training of the model. A batch size of 32 is selected for smooth and balance training. The model is trained for 15 epochs using the Adam optimizer. Adam works well because it adjusts the learning rate and makes gradient updates more efficient.

The results showed a significant improvement in all performance metrics, specifically in recall and F1-score.It indicates that the augmented data effectively enhanced the LSTM model's ability to detect duplicate bug reports. The performance gains validated that data augmentation contributed positively by enriching the training set and reducing class imbalance.

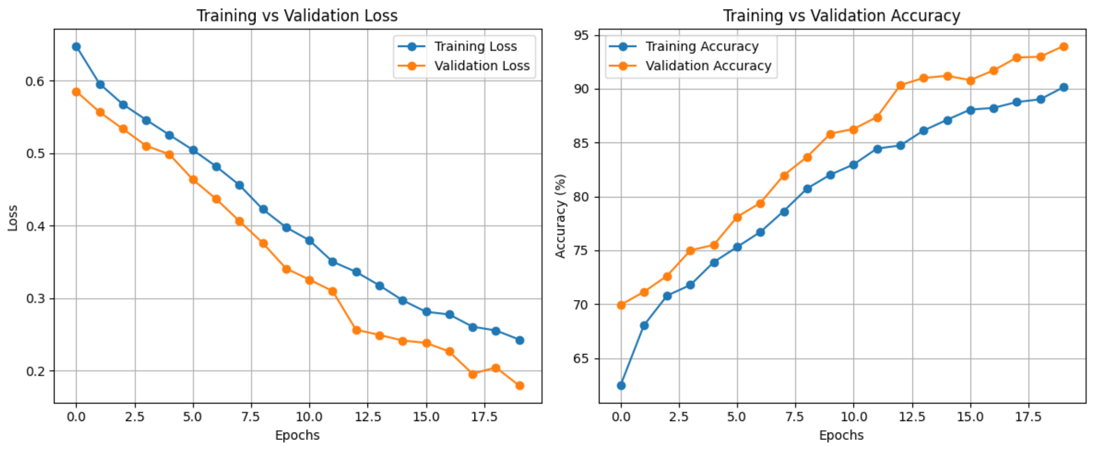


Figure 4.11: LSTM model training vs Validation Accuracy and Loss

Figure 4.11 showed how augmentation resolved the overfitting and performed well.

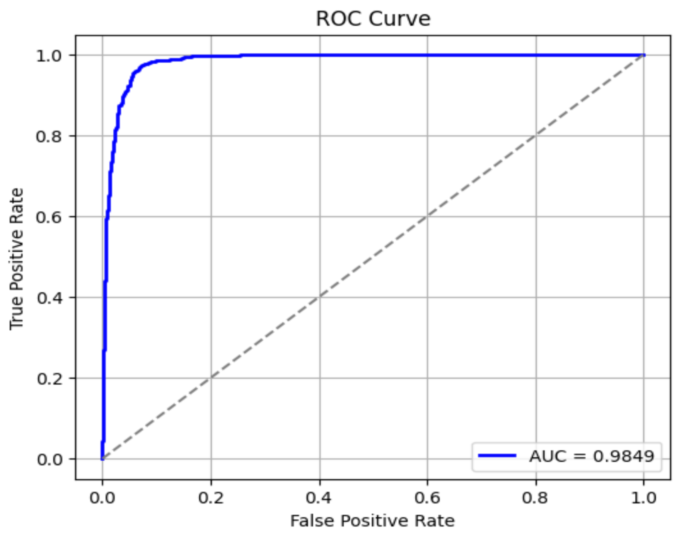


Figure 4.12: LSTM ROC Curve

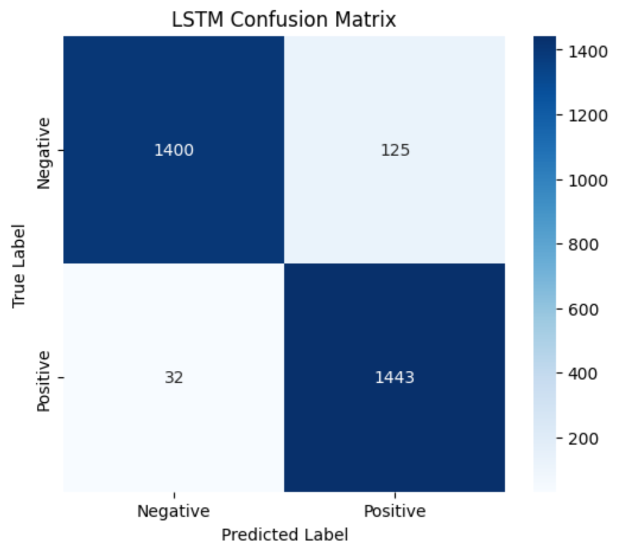


Figure 4.13: LSTM Confusion Matrix

Figure 4.13 explained the model’s performance. It shows model predicted true results.

4.3.2 CNN Model

CNN (Convolutional Neural Network) model is trained on the augmented dataset. To optimize effective feature extraction and classification, we configured the model with hyper parameters. A total of 128 filters are used to capture a wide range of n-gram patterns from the input text. The kernel size of 3 is selected. It enables the model to learn localized semantic features. We implemented 3 convolutional layers, which progressively extracted hierarchical features from the textual input.

A dropout rate of 0.4 is applied across both convolutional and dense layers to improve generalization and prevent overfitting. The learning rate is fixed at 0.0005. It helped the model data training smooth and stable. A batch size of 64 was selected to accelerate the training process while maintaining performance stability.

In the fully connected part of the model, we used several dense layers with decreasing sizes. This helped to compress the features extracted earlier before making the final classification. We used the Adam optimizer along with weight decay. This helped prevent overfitting and made the model generalize better to new data.

The CNN model showed much better results on the augmented data. This was clear from the higher recall and F1-score. It shows that using synthetic data can really help improve deep learning performance.

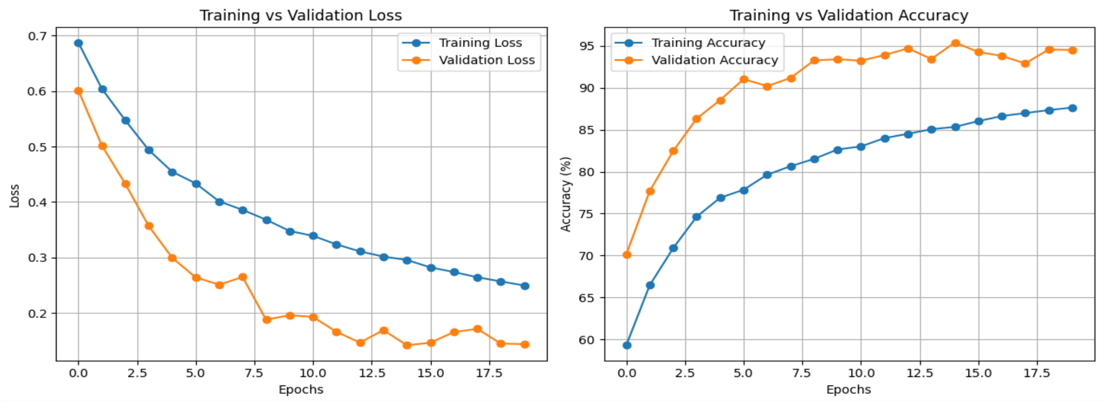


Figure 4.14: CNN model training vs Validation Accuracy and Loss

Figure 4.14 shows how CNN improved performance after resolving class imbalance problem through data augmentation.

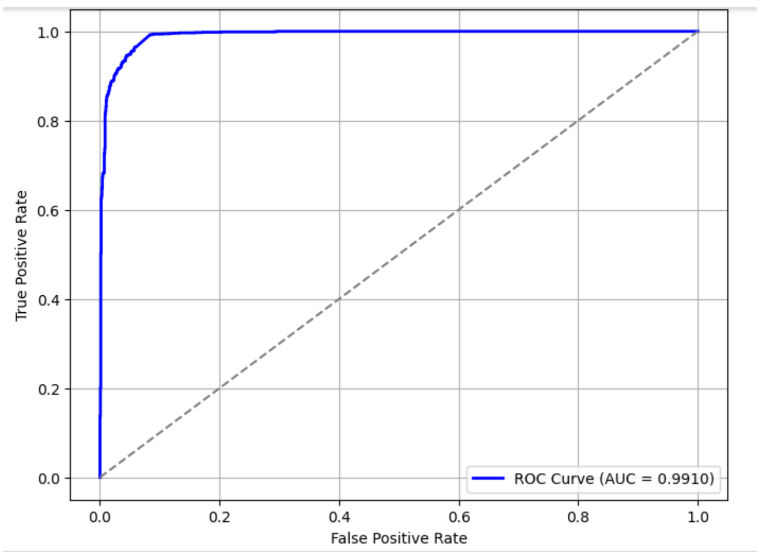


Figure 4.15: CNN ROC Curve

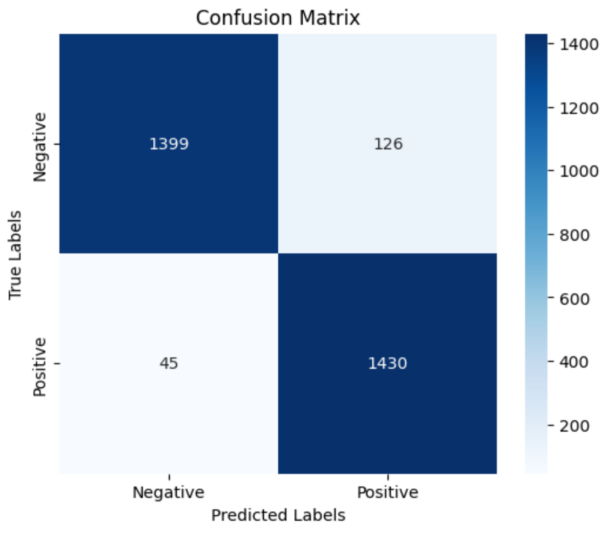


Figure 4.16: CNN Confusion Matrix

Figure 4.16 showed the result of model performance. Model performance improved due to augmentation.

4.3.3 BERT Model

BERT model is trained on the augmented dataset. We applied the same hyper parameters used in the non-augmented setting to ensure consistency and to effectively assess the impact of data augmentation. Specifically, we used the pre-trained 'BERT-base-uncased' model with a learning rate of 2e-5, a batch size of 8, and a dropout rate of 0.3. The AdamW optimizer was employed for its regularization capabilities, and the hidden layer structure remained 768, 256, and 2.

Despite no changes in model configuration, the BERT model achieved significantly higher performance on the augmented dataset across all evaluation metrics. This underscores the strength of the data augmentation techniques applied, which enriched the training data with more diverse and representative examples. The results highlight BERT’s robustness and ability to leverage augmented textual data effectively for improved duplicate bug report detection.

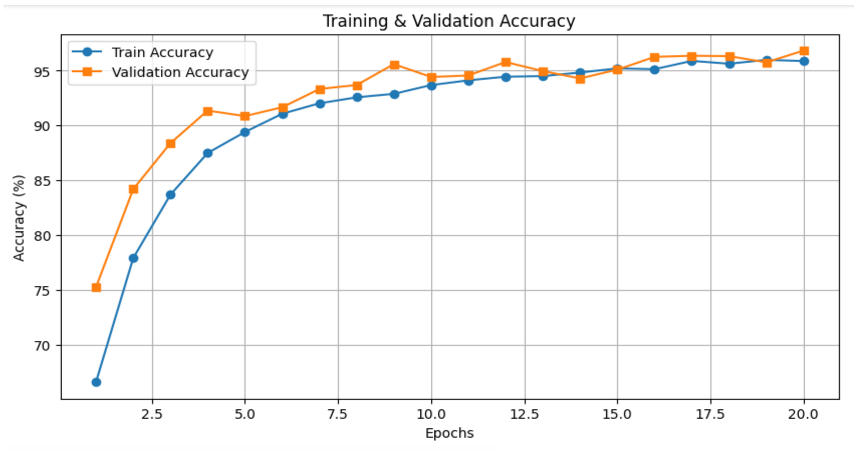


Figure 4.17: BERT Training vs Validation Accuracy

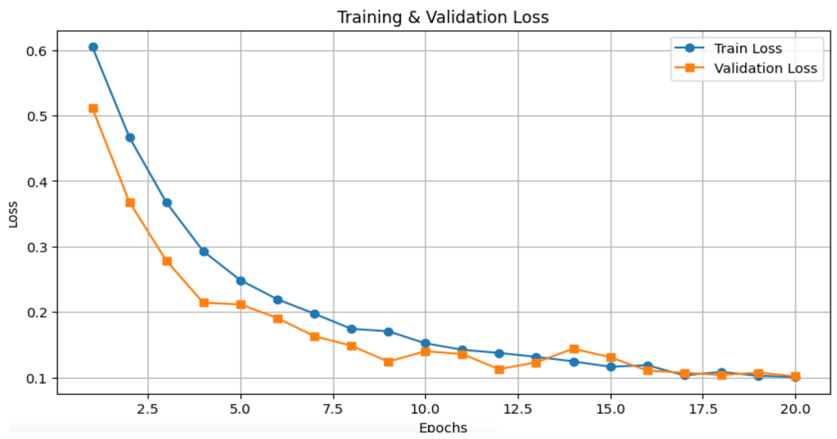


Figure 4.18: BERT Training vs Validation Loss

Figure 4.17 and 4.18 showing how data set trained very well. Model is generalized.

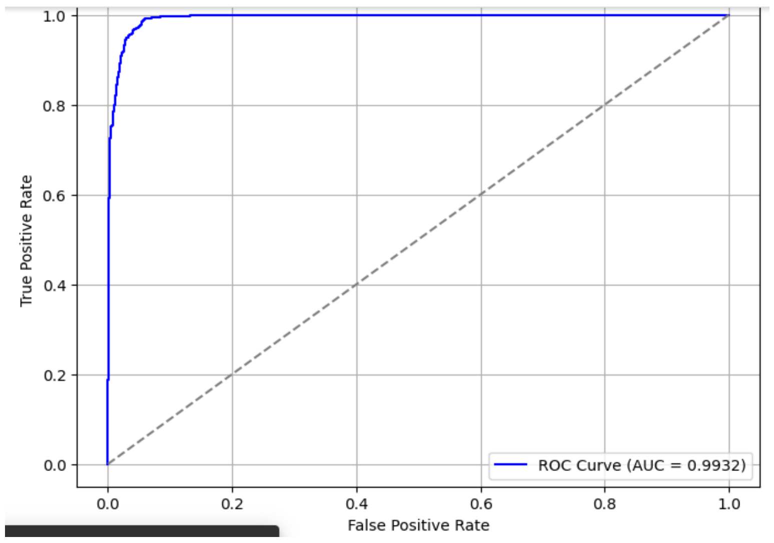


Figure 4.19: BERT ROC Curve

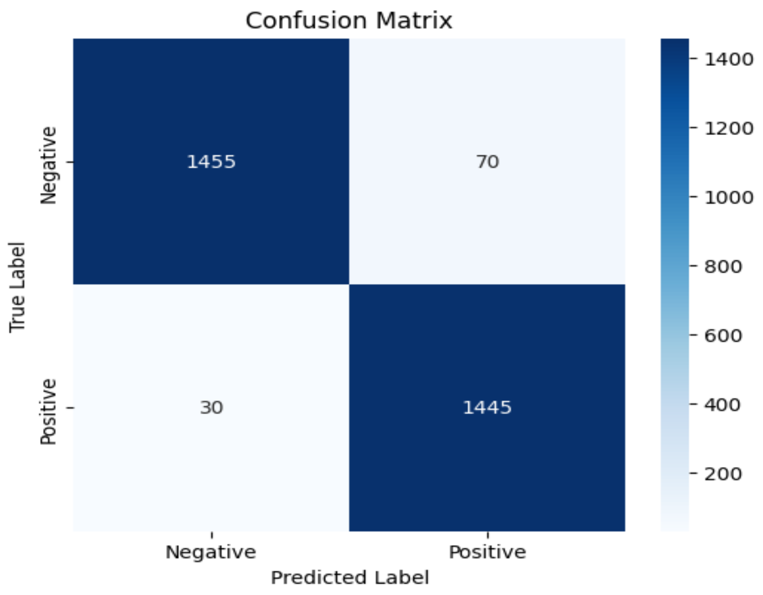


Figure 4.20: BERT Confusion Matrix

Figure 4.20 showing BERT model performance and it showed good results.

4.4 Comparison of Result, Augmented and Non-Augmented

According to results table 4.3, we can see that augmentation improved the accuracy of machine learning models. To verify that augmented data improved the model's performance we decided to check three deep learning models: CNN, LSTM and BERT. CNN, LSTM and BERT across all three models, the application of data augmentation resulted in considerable performance improvements. The enhancements were not only reflected in the accuracy values but were more pronounced in recall and F1-score. These two metrics especially important in class-imbalanced classification problems. The original dataset contained a significantly lower number of duplicate bug reports compared to non-duplicates. As a result, models trained on the non-augmented dataset tended to become biased toward the majority class, often failing to detect duplicates.

Data augmentation helped address this imbalance by generating more representative samples of the minority class. Using techniques such as synonym replacement, random insertion, sentence shuffling, and component reordering, the training set was effectively expanded. These artificially generated samples introduced linguistic variety while preserving the core semantic intent of the reports. This variety made the training process more robust by helping the models recognize duplicate reports that may be phrased differently but refer to the same issue.

The overall improvements across all performance metrics after augmentation support the conclusion that the technique helped in achieving better generalization and reduced overfitting. This was particularly vital for deep learning models, which require a large amount of data to reach optimal performance. Table 4.3 explained the detailed comparison between augmented and non-augmented data set. How following evaluation metrics improved their results on augmented data as compared to non-augmented data.

Table 4.3: Comparisons of Result

Model	Data Type	Accuracy	Precision	Recall	F1-Score	AUC Score
LSTM	Non-Augmented	88.47%	0.9039	0.9308	0.9172	0.97
	Augmented	94.77%	0.9190	0.9695	0.9436	0.98
CNN	Non-Augmented	89.35%	0.4000	0.0732	0.1237	0.69
	Augmented	94.77%	0.9083	0.9939	0.9492	0.99
BERT	Non-Augmented	89.72%	0.5000	0.0366	0.0682	0.68
	Augmented	96.30%	0.9528	0.9715	0.9621	0.99

4.4.1 LSTM Model

Before augmentation: The LSTM model already demonstrated relatively high performance before augmentation, achieving an accuracy of 88.47% and an F1-score of 0.9172. The recall value was particularly impressive at 0.9308, indicating the model was good at correctly identifying actual duplicates. However, this performance likely benefited from the sequential modeling capabilities inherent in LSTM architecture, which is adept at handling textual data with dependencies spread across multiple words.

LSTM networks are particularly strong at learning from ordered data sequences. In the context of bug reports, the order in which terms appear can provide crucial contextual information, especially for understanding how one issue may resemble another. Despite the small dataset, the LSTM model's internal memory structure helped maintain context and sequence, allowing it to make reasonably accurate classifications.

After augmentation: After training on the augmented dataset, the LSTM model showed notable improvements. Accuracy increased to 94.77%, and the F1-score rose to 0.9436. These gains suggest that the model benefited from both a larger and a more balanced training dataset. The improvements in recall and precision (both over 0.91) signify better identification of true positives and fewer false positives and false negatives.

The richer dataset provided more varied sequences for the model to learn from, enabling it to generalize better across unseen bug reports. The new examples introduced via augmentation made the model less likely to over fit and more likely to distinguish subtle differences and similarities between different bug reports, which is key in real-world bug tracking systems.

4.4.2 CNN Model

Before augmentation: The CNN model presented an interesting case. It achieved a relatively high accuracy of 89.35%, but this was misleading. The precision, recall, and F1-score were extremely low, with recall at only 0.0732 and F1-score at 0.1237. These values clearly indicated that the model was failing to identify duplicates effectively and was biased towards predicting the majority class. This is a classic example of accuracy paradox in imbalanced classification problems: high accuracy does not always indicate good performance if minority classes are ignored.

CNNs, while strong in image recognition tasks, often underperform in NLP problems with short, context-sensitive texts unless the dataset is sufficiently large. The model failed to capture the semantic similarities between differently worded duplicate bug reports due to limited and skewed training data.

After augmentation: The augmented dataset brought about a dramatic transformation in CNN's performance. Accuracy remained high at 94.77%, but the real gains were in recall (0.9939) and F1-score (0.9492), showing that the model had learned to detect duplicates with much greater accuracy. Precision also improved significantly to 0.9083.

These improvements suggest that CNNs can perform well in NLP tasks when supported by a large and balanced dataset. The convolutional layers were able to pick up on more diverse patterns and relationships in the text, which were only made available through the use of augmentation techniques. Importantly, this case underscores the necessity of dataset enhancement for CNN-based text classification tasks, particularly when dealing with limited and imbalanced data.

4.4.3 BERT Model

Before augmentation: An accuracy of 89.72% has been shown by the pre-trained BERT model. However, a lower rate of 0.0366 and 0.0682 has been shown by Recall and F1-score. Despite the accuracy, this highlighted an inadequate ability to detect bugs. The fine-turning process is a major reason behind this underperformance.

BERT is an intricate model consisting of a number of parameters. Sufficient and balanced data is required when fine-turning BERT for a specific task, although it comes pre-trained on vast text corpora. BERT generalizes poorly for minority class as a result of the limited and imbalanced data, which leads to low recall and precision for duplicates.

After augmentation: BERT surpassed all other models across every metric after the augmentation was applied. It achieved the highest accuracy at 96.30%, and most impressively, an F1-score of 0.9621, with recall and precision above 0.95. The full potential of the model is highlighted upon given the enough and representative data.

The relationship between different textual descriptions is understood efficiently by BERT with augmented date. The context could be analyzed by it better than CNN and LSTM due to its deep architecture, which resulted in accurate classification even with different structure of duplicate reports.

The augmented dataset gave BERT the linguistic variety it needed to fine-tune effectively, proving that while BERT can be resource-intensive, it offers superior results when appropriately supported by data augmentation.

4.5 Result comparison with existing studies

To prove the effectiveness of study, the results are compared with the state of art approaches of duplicate bug report detection. Table 4.4 shows the comparison of our augmentation base approach with the state of art approaches.

Table 4.4: Comparison with existing studies

Author/Year	Dataset	Techniques	Result
Clare et al. (2025) [67]	Eclipse (Bugzilla repository)	Multi-layer perception neural network (MLP) Naïve Bayes Model, BERT Model	26% precision on duplicate bug report
Q Meng et al. (2024) [61]	Eclipse, Firefox, Mozilla, JDT, Thunder Bird	Bi-LSTM, DC-CNN and Transformer base models (BERT, ALBERT, RoBERTa)	87% Average F1-Score
M. B. Messaoud et al. (2023) [60]	Thunderbird	BERT-MLP using BERT Comparison between (DC-CNN, Sentence BERT)	89.03%
Xiaoxue Wu et al. (2023) [64]	Mozilla Core, Mozilla Firefox, Thunderbird, Eclipse	CTEDB (Combination of Term Extraction and DeBERTaV3) techniques	95% accuracy
Proposed approach	Mobile dataset	Easy data augmentation technique, BERT Model	96% accuracy, 97% recall

In the previous studies they used different machine learning techniques that performed well when they used large data set. In the given table, Meng et al. used Bi-LSTM, DC-CNN and Transformer base models and achieved 87% result because they used large number of data set of different projects. Messaoud et al. proposed another techniques BERT-MLP using BERT Comparison between (DC-CNN, Sentence BERT) and achieved 89.03% accuracy. All these approaches performed well because they used large datasets. There are some studies where the small datasets are used for duplicate bug report detection. They also use machine learning and deep learning approaches but these studies achieved less accuracy. For example, Clare et al. used Multi-layer perception neural network (MLP) Naïve Bayes Model to identify duplicate bug report and achieved 26% accuracy on duplicate bug report because of small and imbalance dataset. The results of the studies as shown in the table depicts that when there is small dataset for training, the machine learning models struggle to achieve the high accuracy. To resolve this issue, our proposed augmentation base approach played a significant role to achieve the high accuracy of deep learning model. BERT model achieved 96% accuracy and 97% recall and overall AUC score is 99%.

4.6 Discussion

By addressing one of the common challenges in machine learning that is the scarcity and imbalance of labeled data, the focus of this research is to focus on increasing the performance of deep learning models for duplicate bug report detection. The number of bug reports available for training is limited for numerous software projects. This limited availability challenges the deep learning models which often require vast volumes of diverse and well-labelled data to achieve generalized performance. Moreover, the number of non-duplicate reports are often higher in number than duplicates which results in imbalanced bug report datasets. This imbalance results in the biasness of the model towards the majority class, which eventually leads to poor detection of duplicate bug reports.

To overcome these challenges, this study employed data augmentation techniques aimed at increasing both the size and diversity of the training data. Data augmentation, a widely recognized approach in machine learning, artificially expands a dataset by creating new samples through transformations applied to existing data. In the context of natural language processing (NLP) and specifically for bug report texts, augmentation is less straightforward compared to images due to the semantic sensitivity of textual data. Therefore, careful selection of augmentation techniques is crucial to avoid generating irrelevant or misleading examples.

Numerous augmentation methods, such as random insertion, synonym replacement, random shuffling, and component shuffling, were applied in this study. The alteration of words by inserting new words for the purpose of diversification of data comes under random insertion. In order to introduce lexical diversity, synonym replacement is used for the purpose of swapping certain words with their synonyms. The fixed word sequence is overcome through the use random shuffling that rearranges words or phrases within the text, which in turn encourages the model to reply on overall semantics rather than specific lexical arrangement. Component shuffling reorders distinct sections of a bug report, such as steps to reproduce or expected behavior, fostering robustness against variations in report structure.

By introducing meaningful variation and training samples, the dataset was enhanced by these augmentation techniques collectively. As learning models, like CNN, LSTM, and BERT reply on varied and diverse data, this expansion is quite crucial. By generating more duplicate bug reports, augmentation reduced the class imbalance issue. This balanced the dataset and prevented the biasness of the models towards the majority of non-duplicate class.

After augmenting the dataset, we retrained three distinct models: Convolutional Neural Network (CNN), Long Short-Term Memory (LSTM), and Bidirectional Encoder Representations from Transformers (BERT). The models were evaluated using multiple metrics, including accuracy, precision, recall, F1-score, and Area Under the Curve (AUC), to provide a comprehensive understanding of their performance improvements.

The results clearly demonstrated that all three models benefited significantly from data augmentation, with improvements seen across all metrics. The BERT model exhibited the most substantial gains, achieving an accuracy of 96.30% and an F1-score of 0.9621, considerably outperforming its non-augmented counterpart and the other models. This is attributed to BERT’s architecture, which uses a transformer-based, bidirectional mechanism capable of understanding context in both directions within a sentence or document. BERT is pre-trained on massive corpora and fine-tuned on the specific bug report data, making it especially effective at capturing nuanced semantic relationships when trained on a large and varied dataset.

The CNN model showed the most dramatic improvement post-augmentation. Initially, CNN struggled with the dataset due to its small size and imbalance, reflected by its low recall (0.0732) and F1-score (0.1237) on non-augmented data. However, after augmentation, the CNN's recall skyrocketed to 0.9939, and its F1-score reached 0.9492. This indicates that CNN's convolutional filters could effectively extract local textual features related to duplicates once provided with a balanced and diverse training set. CNN's strength in recognizing patterns in short text snippets was unlocked by augmentation, highlighting the importance of dataset quality for model performance.

The LSTM model, known for its ability to process sequential data and capture temporal dependencies, performed reasonably well even before augmentation, with an accuracy of 88.47% and a recall of 0.9308. Nonetheless, its performance improved notably with augmentation, achieving 94.77% accuracy and a balanced F1-score of 0.9436. The LSTM's ability to handle variable-length sequences made it robust in learning from the augmented, varied bug report data, enhancing its predictive capacity for duplicates.

Overall, data augmentation served two crucial roles. First, it increased the volume of training data, which is essential for the high capacity of deep learning models to learn effectively without overfitting. Second, it enhanced the diversity and balance of the dataset, which enabled models to learn richer representations and make more accurate predictions for both majority and minority classes.

The success of this approach has important practical implications. It has been compared with existing start of art to prove the success of this approach. In real-world software development environments, acquiring large labeled datasets is often expensive and time-consuming. Manual annotation of duplicate bug reports requires expert knowledge and is prone to inconsistency. Data augmentation thus provides a cost-effective alternative for boosting model performance without the need for extensive new data collection.

Furthermore, this study reinforces the idea that the combination of augmentation and sophisticated models like BERT can create powerful solutions for text classification problems involving limited and imbalanced datasets. While BERT's pre-trained knowledge base is valuable, its performance in domain-specific tasks heavily depends on the quality and quantity of fine-tuning data. Augmentation can therefore be seen as an indispensable step to unlock the full potential of such models.

There are numerous avenues that are worth looking at for further research. Further improvement can be done on the automated augmentation pipelines that use generate context-aware and semantically accurate examples with the help of language models. Additionally, integrating active learning, where models iteratively select the most informative samples for human annotation, can synergize well with augmentation to optimize dataset quality. Finally, deploying these models in real bug tracking systems and evaluating their impact on developer productivity and bug resolution times would provide valuable insights into practical usability.

In conclusion, the fact, that data augmentation can be an effective strategy used to improve the performance of deep learning models on small and imbalanced datasets, is posited by this study. Augmentation reduces model bias by increasing dataset size, which results in improved generalization. This leads to reliable and accurate duplicate bug report detection. The findings highlight the critical role of data quality in machine learning and offer a promising pathway for practical implementations in software engineering domains.

4.7 Summary

A comprehensive assessment of the results taken from the experimental evaluation is provided by this chapter. The proposed approach is used for the basis of the result, whereas its implementation is based on the selected dataset. The main focus is on the accuracy and overall efficiency of the approach that has been used. It goes into great detail on the key outcomes and performance indicators such as precision, recall, F1-score, and AUC score, showing how they can be utilized to enhance the accuracy of duplicate bug report identification in real-world applications. A critical and thorough review is provided by this chapter of the prior studies. The proposed approach has been compared with existing literature. The comparison is primarily based on statistical performance metrics like accuracy, precision, and recall rates. Additionally, the analysis highlights specific strengths, improvements, and innovations introduced by the proposed method. In the scenarios with small datasets or imbalanced data, where traditional models often fail to perform well, these improvement are evident. Overall, this chapter serves to validate the proposed method's superiority and its contribution to the field of software maintenance and bug tracking.

CHAPTER 5

CONCLUSION AND FUTURE WORK

5.1 Conclusion

The identification of duplicate bug reports that is essential for efficient software maintenance. Repetitive task and speed up issue resolution can be solved through it. Due to unavailability of large label dataset, machine learning and deep learning models often struggle to predict accurate result. It may result in poor model outcomes due to limited training data. The motivation behind this research is to investigate techniques that improve the accuracy of these deep learning models that struggle in predicting accurate result especially when historical bug report data is insufficient.

The main research question (RQ1) examined the effectiveness of data augmentation technique that explored in this thesis. It is explored that data augmentation affects the accuracy of machine learning and deep learning methods for detecting duplicate bug reports. The experiment in this research revealed that how LSTM, CNN, and BERT models enhance the performance on trained augmented datasets. Models, such as CNN and BERT, demonstrated low recall and F-1 scores. Therefore, struggling to highlight semantic similarities between duplicate reports. However, all the models showed improvements in key measures upon the application of the augmentation techniques that expanded the training data.

BERT model, with its F1-score rising from 0.0682 to 0.9621 and accuracy increasing from 89.72% to 96.30%, showed notable improvement outperforming other architectures. Similarly, LSTM and CNN models significant improved in precision, recall, and AUC scores with augmented data. This highlights that the small datasets issues can be overcome with augmentation. The possibility of the transformation of poorly performing models into reliable tools is evident in this study through augmentation.

This research presents data augmentation as a powerful approach that can be used to boost the accuracy of models that are used for detecting duplicate bug reports. The answer to RQ1 is directly provided by it, and it asserts machine learning models can be improved by increasing the data size. Future research could build on these insights by investigating domain-specific augmentation methods, incorporating multi-modal data such as logs or screenshots, and optimizing transformer-based models for broader use in real-world software development contexts.

5.2 Future Work

For Future work, I purpose that we can explore more targeted augmentation techniques for technical text. We are able to analyze the impact of different augmentation ratios to evaluate the performance of machine learning models. We can use different types of data like stack traces, screenshots, and logs to improve duplicate bug report detection. These extra details can give more clues about the bug and make it easier to find similar reports. Using multimodal data can help the model understand the bug more clearly and make better predictions. Leveraging larger pre-trained language models like GPT or RoBERTa could improve contextual understanding. Developing real time duplicate detection systems for integration into issue tracking tools is another promising direction. We can experiment with different approaches like few-shot and zero-shot learning approaches. It can give benefit projects with extremely limited data. Cross-project transfer learning can help models generalize better across different software domains.

References

[1] J. Zou, L. Xu, M. Yang, M. Yan, D. Yang, and X. Zhang, “Duplication Detection for Software Bug Reports based on Topic Model,” in *2016 9th International Conference on Service Science (ICSS)*, 2016, pp. 60–65. doi: 10.1109/ICSS.2016.16.

[2] A. Hindle and C. Onuczko, “Preventing duplicate bug reports by continuously querying bug reports,” *Empir Softw Eng*, vol. 24, no. 2, pp. 902–936, Apr. 2019, doi: 10.1007/s10664-018-9643-4.

[3] J. W. Zhang, L. Xiao, M. J. Li, Z. R. Meng, and Y. H. Li, “HYDBre: A Hybrid Retrieval Method for Detecting Duplicate Software Bug Reports,” in *Proceedings - 2024 11th International Conference on Dependable Systems and Their Applications, DSA 2024*, Institute of Electrical and Electronics Engineers Inc., 2024, pp. 242–251. doi: 10.1109/DSA63982.2024.00040.

[4] Y. Wang, Q. Yao, J. T. Kwok, and L. M. Ni, “Generalizing from a Few Examples,” *ACM Computing Surveys (CSUR)*, vol. 53, no. 3, Jun. 2020, doi: 10.1145/3386252.

[5] T. Zhang *et al.*, “Duplicate Bug Report Detection: How Far Are We?,” *ACM Transactions on Software Engineering and Methodology*, vol. 32, no. 4, May 2023, doi: 10.1145/3576042.

[6] T. Kim and G. Yang, “Predicting Duplicate in Bug Report Using Topic-Based Duplicate Learning With Fine Tuning-Based BERT Algorithm,” *IEEE Access*, vol. 10, pp. 129666–129675, 2022, doi: 10.1109/ACCESS.2022.3226238.

[7] D. G. Widder and C. Le Goues, “What Is a ‘Bug’?,” Nov. 01, 2024, *Association for Computing Machinery*. doi: 10.1145/3662730.

[8] S. Gupta and S. K. Gupta, "A Systematic Study of Duplicate Bug Report Detection," *International Journal of Advanced Computer Science and Applications*, vol. 12, 2021, [Online]. Available: https://api.semanticscholar.org/CorpusID:232167864

[9] N. Fatima, "Duplicate Bug Report Detection Using Hybrid Model," 2023.

[10] A. Sureka and P. Jalote, "Detecting duplicate bug report using character N-gram-based features," in *Proceedings - Asia-Pacific Software Engineering Conference, APSEC*, 2010, pp. 366–374. doi: 10.1109/APSEC.2010.49.

[11] E. Shihab *et al.*, "Studying re-opened bugs in open source software," *Empir Softw Eng*, vol. 18, no. 5, pp. 1005–1042, Oct. 2013, doi: 10.1007/s10664-012-9228-6.

[12] L. Ghadhab, I. Jenhani, M. W. Mkaouer, and M. Ben Messaoud, "Augmenting commit classification by using fine-grained source code changes and a pre-trained deep neural language model," *Inf Softw Technol*, vol. 135, p. 106566, Jul. 2021, doi: 10.1016/J.INFSOF.2021.106566.

[13] G. Yang, J. Ji, and T. Kim, "Feature Learning via Correlation Analysis for Effective Duplicate Detection," *Applied Sciences (Switzerland)*, vol. 15, no. 3, Feb. 2025, doi: 10.3390/app15031411.

[14] P. M. Nadkarni, L. Ohno-Machado, and W. W. Chapman, "Natural language processing: An introduction," Sep. 2011. doi: 10.1136/amiajnl-2011-000464.

[15] J. Camacho-Collados and M. T. Pilehvar, "On the Role of Text Preprocessing in Neural Network Architectures: An Evaluation Study on Text Categorization and Sentiment Analysis," Aug. 2018, [Online]. Available: http://arxiv.org/abs/1707.01780

[16] "Natural Language Processing in Action, Second Edition - Hobson Lane, Maria Dyshel - Google Books." Accessed: Mar. 19, 2025. [Online]. Available: https://books.google.com.pk/books?hl=en&lr=&id=hBRDEQAAQBAJ&oi=fnd&pg=PA1&dq=Natural+language+processing+pipeline&ots=uAmMcZIDp9&sig=mv1H

sceYCUIRnrEdEIzoWiEmPOo&redir_esc=y#v=onepage&q=Natural%20language%20processing%20pipeline&f=false

[17] S. Symeonidis, D. Effrosynidis, and A. Arampatzis, “A comparative evaluation of pre-processing techniques and their interactions for twitter sentiment analysis,” *Expert Syst Appl*, vol. 110, pp. 298–310, 2018, doi: https://doi.org/10.1016/j.eswa.2018.06.022.

[18] R. Pramana, Debora, J. J. Subroto, A. A. S. Gunawan, and Anderies, “Systematic Literature Review of Stemming and Lemmatization Performance for Sentence Similarity,” *Proceedings of the 2022 IEEE 7th International Conference on Information Technology and Digital Applications, ICITDA 2022*, 2022, doi: 10.1109/ICITDA55840.2022.9971451.

[19] S. Garg and A. Garg, “Comparison of machine learning algorithms for content based personality resolution of tweets,” *Social Sciences & Humanities Open*, vol. 4, no. 1, p. 100178, Jan. 2021, doi: 10.1016/J.SSAHO.2021.100178.

[20] S. Qiu, Q. Liu, S. Zhou, and W. Huang, “Adversarial attack and defense technologies in natural language processing: A survey,” *Neurocomputing*, vol. 492, pp. 278–307, Jul. 2022, doi: 10.1016/j.neucom.2022.04.020.

[21] M. and X. D. and G. J. and W. F. and Z. L. Li Ruiguang and Liu, “A Review of Machine Learning Algorithms for Text Classification,” in *Cyber Security*, Y. and W. W. and Y. H. and L. C. Lu Wei and Zhang, Ed., Singapore: Springer Nature Singapore, 2022, pp. 226–234.

[22] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding,” May 2019, [Online]. Available: http://arxiv.org/abs/1810.04805

[23] M. Hossin and Sulaiman, “(),” *International Journal of Data Mining & Knowledge Management Process (IJDKP)*, vol. 5, no. 2, 2015, doi: 10.5121/ijdkp.2015.5201.

[24] K. M. B. M. B. Mohammed M, *Machine Learning: Algorithms and Applications*. 2016.

[25] H. Skogström, “Blog / DLOps: MLOps for Deep Learning.”

[26] C. Janiesch, P. Zschech, and K. Heinrich, “Machine learning and deep learning”, doi: 10.1007/s12525-021-00475-2/Published.

[27] I. H. Sarker, “Machine Learning: Algorithms, Real-World Applications and Research Directions,” May 01, 2021, *Springer*. doi: 10.1007/s42979-021-00592-x.

[28] V. Dogra *et al.*, “A Complete Process of Text Classification System Using State-of-the-Art NLP Models,” 2022, *Hindawi Limited*. doi: 10.1155/2022/1883698.

[29] Z. Xu, M. Chen, K. Q. Weinberger, and F. Sha, “An alternative text representation to TF-IDF and Bag-of-Words *”.

[30] K. Das, Kamlish, and F. Abid, “Advancements in Word Embeddings: A Comprehensive Survey and Analysis,” *Proceedings of the Pakistan Academy of Sciences: A. Physical and Computational Sciences*, vol. 61, no. 3, pp. 227–245, Sep. 2024, doi: 10.53560/PPASA(61-3)842.

[31] G. Bonisoli, F. Rollo, and L. Po, “Using Word Embeddings for Italian Crime News Categorization,” *Proceedings of the 16th Conference on Computer Science and Intelligence Systems, FedCSIS 2021*, pp. 461–470, Sep. 2021, doi: 10.15439/2021F118.

[32] T. Mikolov, K. Chen, G. Corrado, and J. Dean, “Efficient Estimation of Word Representations in Vector Space”, Accessed: Aug. 25, 2025. [Online]. Available: http://ronan.collobert.com/senna/

[33] Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning,” *Nature*, vol. 521, no. 7553, pp. 436–444, May 2015, doi: 10.1038/nature14539.

[34] "What is a Convolutional Neural Network? An Engineer's Guide." Accessed: Apr. 04, 2025. [Online]. Available: https://zilliz.com/glossary/convolutional-neural-network

[35] Y. Zhang and B. C. Wallace, "A Sensitivity Analysis of (and Practitioners' Guide to) Convolutional Neural Networks for Sentence Classification," Oct. 2015, Accessed: Apr. 10, 2025. [Online]. Available: https://arxiv.org/abs/1510.03820v4

[36] H. Salehinejad, S. Sankar, J. Barfett, E. Colak, and S. Valaee, "Recent Advances in Recurrent Neural Networks," Dec. 2017, Accessed: Apr. 10, 2025. [Online]. Available: https://arxiv.org/abs/1801.01078v3

[37] "Implementation of RNN, LSTM, and GRU | by Chandra Churh Chatterjee | TDS Archive | Medium." Accessed: Apr. 04, 2025. [Online]. Available: https://medium.com/data-science/implementation-of-rnn-lstm-and-gru-a4250bf6c090

[38] K. Smagulova and A. P. James, "A survey on LSTM memristive neural network architectures and applications," *Eur. Phys. J. Special Topics*, vol. 228, pp. 2313–2324, 2019, doi: 10.1140/epjst/e2019-900046-x.

[39] C. W. Chen, S. P. Tseng, T. W. Kuan, and J. F. Wang, "Outpatient Text Classification Using Attention-Based Bidirectional LSTM for Robot-Assisted Servicing in Hospital," *Information 2020, Vol. 11, Page 106*, vol. 11, no. 2, p. 106, Feb. 2020, doi: 10.3390/INFO11020106.

[40] S. Harrer, "Attention is not all you need: the complicated case of ethically using large language models in healthcare and medicine Personal View," 2023. [Online]. Available: www.thelancet.com

[41] M. Schrimpf and I. Asher Blank, "The neural architecture of language: Integrative modeling converges on predictive processing," vol. 118, no. 45, pp. 1–12, Nov. 2021.

[42] "The Illustrated BERT, ELMo, and co. (How NLP Cracked Transfer Learning) – Jay Alammar – Visualizing machine learning one concept at a time." Accessed: Mar. 23, 2025. [Online]. Available: https://jalammar.github.io/illustrated-bert/

[43] J. Devlin, M. W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," *NAACL HLT 2019 - 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies - Proceedings of the Conference*, vol. 1, pp. 4171–4186, Oct. 2018, Accessed: Aug. 25, 2025. [Online]. Available: https://arxiv.org/pdf/1810.04805

[44] "A Tutorial on using BERT for Text Classification w Fine Tuning." Accessed: Mar. 23, 2025. [Online]. Available: https://pysnacks.com/machine-learning/bert-text-classification-with-fine-tuning/

[45] M. Bilal and A. A. Almazroi, "Effectiveness of Fine-tuned BERT Model in Classification of Helpful and Unhelpful Online Customer Reviews," *Electronic Commerce Research*, vol. 23, no. 4, pp. 2737–2757, Dec. 2023, doi: 10.1007/s10660-022-09560-w.

[46] C. Sun, X. Qiu, Y. Xu, and X. Huang, "How to Fine-Tune BERT for Text Classification?," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 11856 LNAI, pp. 194–206, May 2019, doi: 10.1007/978-3-030-32381-3_16.

[47] S. El Anigri, M. M. Himmi, and A. Mahmoudi, "How BERT's Dropout Fine-Tuning Affects Text Classification?," *Lecture Notes in Business Information Processing*, vol. 416 LNBIP, pp. 130–139, 2021, doi: 10.1007/978-3-030-76508-8_11.

[48] I. Salah, K. Jouini, and O. Korbaa, "On the use of text augmentation for stance and fake news detection," *Journal of Information and Telecommunication*, vol. 7, no. 3, pp. 359–375, 2023, doi: 10.1080/24751839.2023.2198820.

[49] K. Nanthini, D. Sivabalaselvamani, K. Chitra, P. Gokul, S. KavinKumar, and S. Kishore, “A Survey on Data Augmentation Techniques,” in *Proceedings - 7th International Conference on Computing Methodologies and Communication, ICCMC 2023,* Institute of Electrical and Electronics Engineers Inc., 2023, pp. 913–920. doi: 10.1109/ICCMC56507.2023.10084010.

[50] J. Wei and K. Zou, “EDA: Easy Data Augmentation Techniques for Boosting Performance on Text Classification Tasks,” Jan. 2019, [Online]. Available: http://arxiv.org/abs/1901.11196

[51] S. Y. Feng *et al.*, “A Survey of Data Augmentation Approaches for NLP,” *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pp. 968–988, 2021, doi: 10.18653/V1/2021.FINDINGS-ACL.84.

[52] “TextAttack: Text Data Augmentation in NLP - Analytics Vidhya.” Accessed: Mar. 24, 2025. [Online]. Available: https://www.analyticsvidhya.com/blog/2022/02/text-data-augmentation-in-natural-language-processing-with-texattack/

[53] J. Wei and K. Zou, “EDA: Easy Data Augmentation Techniques for Boosting Performance on Text Classification Tasks,” *EMNLP-IJCNLP 2019 - 2019 Conference on Empirical Methods in Natural Language Processing and 9th International Joint Conference on Natural Language Processing, Proceedings of the Conference*, pp. 6382–6388, 2019, doi: 10.18653/V1/D19-1670.

[54] *2019 5th International Conference on Web Research (ICWR) : Tehran, Iran, April 24-25, 2019*. IEEE, 2019.

[55] Q. Xie, Z. Wen, J. Zhu, C. Gao, and Z. Zheng, “Detecting Duplicate Bug Reports with Convolutional Neural Networks,” in *Proceedings - Asia-Pacific Software Engineering Conference, APSEC*, IEEE Computer Society, Jul. 2018, pp. 416–425. doi: 10.1109/APSEC.2018.00056.

[56] A. Kukkar, R. Mohana, Y. Kumar, A. Nayyar, M. Bilal, and K. S. Kwak, “Duplicate Bug Report Detection and Classification System Based on Deep Learning Technique,” *IEEE Access*, vol. 8, pp. 200749–200763, 2020, doi: 10.1109/ACCESS.2020.3033045.

[57] J. He, L. Xu, M. Yan, X. Xia, and Y. Lei, “Duplicate bug report detection using dual-channel convolutional neural networks,” in *IEEE International Conference on Program Comprehension*, IEEE Computer Society, Jul. 2020, pp. 117–127. doi: 10.1145/3387904.3389263.

[58] B. Soleimani Neysiani, S. M. Babamir, and M. Aritsugi, “Efficient feature extraction model for validation performance improvement of duplicate bug report detection in software bug triage systems,” *Inf Softw Technol*, vol. 126, p. 106344, Oct. 2020, doi: 10.1016/J.INFSOF.2020.106344.

[59] A. Patil, K. Han, and A. Jadon, “A Comparative Analysis of Text Embedding Models for Bug Report Semantic Similarity,” in *Proceedings - 11th International Conference on Signal Processing and Integrated Networks, SPIN 2024*, Institute of Electrical and Electronics Engineers Inc., 2024, pp. 262–267. doi: 10.1109/SPIN60856.2024.10512000.

[60] M. Ben Messaoud, A. Miladi, I. Jenhani, M. W. Mkaouer, and L. Ghadhab, “Duplicate Bug Report Detection Using an Attention-Based Neural Language Model,” *IEEE Trans Reliab*, vol. 72, no. 2, pp. 846–858, Jun. 2023, doi: 10.1109/TR.2022.3193645.

[61] Q. Meng, X. Zhang, G. Ramackers, and V. Joost, “Combining Retrieval and Classification: Balancing Efficiency and Accuracy in Duplicate Bug Report Detection,” Apr. 2024, [Online]. Available: http://arxiv.org/abs/2404.14877

[62] H. Isotani, H. Washizaki, Y. Fukazawa, T. Nomoto, S. Ouji, and S. Saito, “Duplicate Bug Report Detection by Using Sentence Embedding and Fine-tuning,” in *Proceedings - 2021 IEEE International Conference on Software Maintenance and*

Evolution, ICSME 2021, Institute of Electrical and Electronics Engineers Inc., 2021, pp. 535–544. doi: 10.1109/ICSME52107.2021.00054.

[63] Y. Wang *et al.*, “Sentence embedding and fine-tuning to automatically identify duplicate bugs.”

[64] X. Wu, W. Shan, W. Zheng, Z. Chen, T. Ren, and X. Sun, “An Intelligent Duplicate Bug Report Detection Method Based on Technical Term Extraction,” in *Proceedings - 2023 IEEE/ACM International Conference on Automation of Software Test, AST 2023*, Institute of Electrical and Electronics Engineers Inc., 2023, pp. 1–12. doi: 10.1109/AST58925.2023.00005.

[65] A. Patil, K. Han, and S. Mukhopadhyay, “A Comparative Study of Text Embedding Models for Semantic Text Similarity in Bug Reports,” Aug. 2023, [Online]. Available: http://arxiv.org/abs/2308.09193

[66] M. Götharsson, K. Stahre, G. Gay, and F. G. De Oliveira Neto, “Exploring the Role of Automation in Duplicate Bug Report Detection: An Industrial Case Study,” in *Proceedings - 2024 IEEE/ACM International Conference on Automation of Software Test, AST 2024*, Association for Computing Machinery, Inc, Apr. 2024, pp. 193–203. doi: 10.1145/3644032.3644450.

[67] C. E. Laney, A. Barovic, and A. Moin, “Automated Duplicate Bug Report Detection in Large Open Bug Repositories,” Apr. 2025, [Online]. Available: http://arxiv.org/abs/2504.14797

[68] U. Mukherjee and M. M. Rahman, “Understanding the Impact of Domain Term Explanation on Duplicate Bug Report Detection,” Mar. 2025, [Online]. Available: http://arxiv.org/abs/2503.18832

[69] W. Zheng, Y. Li, X. Wu, and J. Cheng, "Duplicate Bug Report detection using Named Entity Recognition," *Knowl Based Syst*, vol. 284, p. 111258, Jan. 2024, doi: 10.1016/J.KNOSYS.2023.111258.

[70] H. Dai *et al.*, "AugGPT: Leveraging ChatGPT for Text Data Augmentation," Feb. 2023, [Online]. Available: http://arxiv.org/abs/2302.13007

[71] P. Liu, X. Wang, C. Xiang, and W. Meng, "A Survey of Text Data Augmentation," in *Proceedings - 2020 International Conference on Computer Communication and Network Security, CCNS 2020*, Institute of Electrical and Electronics Engineers Inc., Aug. 2020, pp. 191–195. doi: 10.1109/CCNS50731.2020.00049.

[72] J. Wei and K. Zou, "EDA: Easy Data Augmentation Techniques for Boosting Performance on Text Classification Tasks," Jan. 2019, [Online]. Available: http://arxiv.org/abs/1901.11196

[73] T. Zhang, I. C. Irsan, F. Thung, and D. Lo, "Cupid: Leveraging ChatGPT for More Accurate Duplicate Bug Report Detection," Aug. 2023, [Online]. Available: http://arxiv.org/abs/2308.10022

[74] S. Jahan and M. M. Rahman, "Towards Understanding the Impacts of Textual Dissimilarity on Duplicate Bug Report Detection," in *Proceedings - 2023 IEEE International Conference on Software Analysis, Evolution and Reengineering, SANER 2023*, Institute of Electrical and Electronics Engineers Inc., 2023, pp. 25–36. doi: 10.1109/SANER56733.2023.00013.

[75] Sunghun. Kim, Massimiliano. Di Penta, and Thomas. Zimmermann, *2013 10th Working Conference on Mining Software Repositories (MSR) : proceedings, May 18-19, 2013, San Francisco, CA, USA*. IEEE, 2013.

[76] N. Klein, C. S. Corley, and N. A. Kraft, "New features for duplicate bug detection," in *11th Working Conference on Mining Software Repositories, MSR 2014 -*

Proceedings, Association for Computing Machinery, May 2014, pp. 324–327. doi: 10.1145/2597073.2597090.

[77] S. Gupta and S. K. Gupta, "A Systematic Study of Duplicate Bug Report Detection," *International Journal of Advanced Computer Science and Applications*, vol. 12, 2021, [Online]. Available: https://api.semanticscholar.org/CorpusID:232167864

[78] Y. Tian, C. Sun, and D. Lo, "Improved duplicate bug report identification," in *Proceedings of the European Conference on Software Maintenance and Reengineering, CSMR*, 2012, pp. 385–390. doi: 10.1109/CSMR.2012.48.

[79] S. Frühwirth-Schnatter, "DATA AUGMENTATION AND DYNAMIC LINEAR MODELS," *J Time Ser Anal*, vol. 15, no. 2, pp. 183–202, Mar. 1994, doi: 10.1111/J.1467-9892.1994.TB00184.X.

[80] A. R. Nair, R. P. Singh, D. Gupta, and P. Kumar, "Evaluating the Impact of Text Data Augmentation on Text Classification Tasks using DistilBERT," *Procedia Comput Sci*, vol. 235, pp. 102–111, Jan. 2024, doi: 10.1016/J.PROCS.2024.04.013.

[81] Z. Wang *et al.*, "A Comprehensive Survey on Data Augmentation," May 2025, [Online]. Available: http://arxiv.org/abs/2405.09591

[82] A. Karimi, L. Rossi, and A. Prati, "AEDA: An Easier Data Augmentation Technique for Text Classification," Aug. 2021, [Online]. Available: http://arxiv.org/abs/2108.13230

[83] M. Abulaish and A. K. Sah, "A Text Data Augmentation Approach for Improving the Performance of CNN," in *2019 11th International Conference on Communication Systems and Networks, COMSNETS 2019*, Institute of Electrical and Electronics Engineers Inc., May 2019, pp. 625–630. doi: 10.1109/COMSNETS.2019.8711054.

[84] A. K. Nandanwar and J. Choudhary, “Contextual Embeddings-Based Web Page Categorization Using the Fine-Tune BERT Model,” *Symmetry (Basel)*, vol. 15, no. 2, Feb. 2023, doi: 10.3390/sym15020395.