Applicability of Sequential Models in Software Defect Prediction

A DISSERTATION

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

MASTER OF TECHNOLOGY
IN
SOFTWARE ENGINEERING

Submitted by:

ABHISHEK YADAV

2K22/SWE/01

Under the supervision of

Mrs. Priya Singh

Assistant Professor

DEPARTMENT OF SOFTWARE ENGINEERING

DELHI TECHNOLOGICAL UNIVERSITY

(Formerly Delhi College of Engineering)
Bawana Road, Delhi – 110042

MAY, 2024

DEPARTMENT OF SOFTWARE ENGINEERING
DELHI TECHNOLOGICAL UNIVERSITY
(Formerly Delhi College of Engineering)
Bawana Road, Delhi-110042

CANDIDATE'S DECLARATION

I, Abhishek Yadav, 2K22/SWE/01 student of M. Tech (SWE), hereby declare that the project entitled "**Applicability of Sequential Models in Software Defect Prediction**" which is submitted by me to Department of Software Engineering, Delhi Technological University, Shahbad Daulatpur, Delhi in partial fulfilment of requirement for the award of the degree of Master of Technology in Software Engineering, has not been previously formed the basis for any fulfilment of requirement in any degree or other similar title or recognition. This report is an authentic record of my work carried out during my degree under the guidance of **Mrs. Priya Singh.**

Place: Delhi
Date:

Abhishek Yadav
(2K22/SWE/01)

DEPARTMENT OF SOFTWARE ENGINEERING
DELHI TECHNOLOGICAL UNIVERSITY
(Formerly Delhi College of Engineering)
Bawana Road, Delhi-110042

CERTIFICATE

I hereby certify that the project entitled "**Applicability of Sequential Models in Software Defect Prediction**" which is submitted by Abhishek Yadav(2K22/SWE/01) to Department of Software Engineering, Delhi Technological University, Shahbad Daulatpur, Delhi in partial fulfilment of requirement for the award of the degree of Master of Technology in Software Engineering, is a record of the project work carried out by the student under my supervision.

To the best of my knowledge this work has not been submitted in part or full for any degree or diploma to this university or elsewhere.

Place: Delhi

Date:

Mrs. Priya Singh
(Assistant Professor, SE, DTU)
SUPERVISOR

ACKNOWLEDGEMENT

I am very thankful to Mrs. Priya Singh (Assistant Professor, DTU, Department of Software Engineering) and all the faculty members of the Department of Software Engineering at DTU. They all provided us with immense support and guidance for the project. I would also like to express my gratitude to the University for providing us with the laboratories, infrastructure, testing facilities and environment which allowed us to work without any obstructions.

I would also like to appreciate the support provided to us by our lab assistants, seniors and our peer group who aided us with all the knowledge they had regarding various topics.

Abhishek Yadav

(2K22/SWE/01)

M Tech (SWE)

ABSTRACT

This study conducts a comprehensive literature review on Software Defect with a focus on the application of sequential models. Software defects can have severe repercussions, ranging from system failures to security vulnerabilities, necessitating the development of effective defect prediction techniques. While traditional methods have relied on handcrafted features and statistical models, the emergence of sequential models in the form of LSTM, RNN, and GRU, originally designed for sequence data, has garnered attention in the software engineering field. This review aims to provide an in-depth analysis of existing research on the use of sequential models for SDP. The study highlights methodologies, datasets, evaluation metrics, advantages, and challenges encountered in applying LSTM, RNN, and GRU for this purpose. The findings of this review are intended to guide researchers and practitioners in leveraging sequential models to enhance software quality. SDP plays a crucial role in software quality assurance, aiming to identify and address potential issues before they lead to failures or other adverse consequences. Traditional techniques often face limitations in capturing complex relationships between software artifacts and defects. Sequential models have emerged as promising alternatives for defect prediction tasks, showcasing improved performance and adaptability.

This paper provides an introduction to software defect prediction, highlighting its importance and the challenges it poses, along with conventional approaches. We then delve into sequential models, discussing their principles, structure, functionality, and advantages in the context of defect prediction. Our review assesses current research on LSTM, RNN, and GRU-based software defect prediction, examining key contributions, methodologies, and results. We also explore various aspects of sequential models, including data pre-processing, network architecture design, and performance evaluation metrics. Through this analysis, we identify the strengths and limitations of these models, as well as the challenges and open research questions in the field. We conclude by highlighting areas for improvement in terms of interpretability, scalability, and robustness of these models.

Finally, we propose future research directions and potential avenues for advancements in software defect prediction using sequential. These recommendations include the exploration of novel network architectures, the incorporation of domain-specific knowledge, and the development of hybrid models that combine the strengths of both traditional and deep learning techniques. Emphasis is also placed on the need for extensive empirical studies and

benchmarking efforts to facilitate the comparison and evaluation of different sequential model approaches.

TABLE OF CONTENTS

PAGE NO.

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

SDP:	Software Defect Prediction
LSTM:	Long Short-Term Memory
BI-LSTM:	Bidirectional long short-term memory
CBOW:	Continuous Bag of Words
RNN:	Recurrent Neural Network
NLP:	Natural Language Processing

CHAPTER 1

INTRODUCTION

1.1 Overview

Software defects are a common occurrence in the field of software development, and they pose significant challenges to both developers and organizations. These defects can take different forms, ranging from minor bugs to critical vulnerabilities, which can result in system failures, security breaches, and financial losses. Therefore, it is crucial to develop effective defect prediction techniques that can help identify and address these issues proactively.

Traditional approaches to defect prediction often rely on manual inspection, static analysis, and statistical modeling. These methods typically involve the extraction of handcrafted features from software artifacts, such as source code metrics, and the application of machine learning algorithms to classify or predict the presence of defects. While these techniques have been valuable, they may struggle to capture the intricate relationships and patterns present in software code, particularly in large-scale, complex systems.

In recent years, the emergence of deep learning and sequential models has brought about significant changes in various fields, including NLP, computer vision, and time series analysis. Sequential models like LSTM [1][8][20], RNN [10][11][26], and GRU [14][19][28] are specifically created to process sequential data, making them ideal for tasks involving temporal dependencies and context.

In the context of software defect prediction, the application of sequential models represents a paradigm shift from traditional methods. These models offer the potential to capture complex temporal relationships within software artifacts, leading to improved prediction accuracy and generalization. By leveraging the inherent structure and semantics of software code, sequential models have the capability to uncover hidden patterns and nuances that may evade conventional approaches.

1.2 Motivation

The purpose of this review is to improve the methodologies used in SDP, which are becoming more necessary as modern software systems become more complex and larger in scale.

Although traditional techniques can be effective in some situations, they may fall short when dealing with the intricacies of real-world software code. Factors such as the interaction between multiple software components, changing requirements, and rapid development cycles make it challenging to predict defects with accuracy.

Sequential models offer a promising alternative to address these challenges by providing a framework for capturing temporal dependencies and contextual information within software artifacts. The ability of these models to learn from sequential data, such as source code revisions, execution traces, and user interactions, presents an opportunity to uncover latent patterns and correlations that may signify the presence of defects.

By exploring the application of sequential models in SDP, this review aims to bridge the gap between traditional approaches and emerging techniques. By providing insights into the strengths, limitations, and practical considerations of using sequential models for defect prediction, this review seeks to empower researchers and practitioners to make informed decisions and drive advancements in software quality assurance.

1.3 Problem Statement

Despite the potential of sequential models in software defect prediction, there exists a gap in understanding their effectiveness, applicability, and limitations in real-world scenarios. Existing research in this area is fragmented, with studies focusing on specific aspects or applications of sequential models without providing a comprehensive overview of the field. This fragmentation hinders the development of a cohesive understanding of the strengths and weaknesses of sequential models in defect prediction.

Furthermore, the lack of standardized methodologies, evaluation metrics, and benchmark datasets complicates the comparison and reproducibility of results across studies. This variability makes it challenging for researchers and practitioners to assess the true efficacy of sequential models and identify best practices for their implementation.

Therefore, there is a need for a structured review that systematically evaluates the existing literature on the application of sequential models for software defect prediction. Such a review should encompass a broad range of methodologies, datasets, evaluation metrics, and case studies to provide a holistic perspective on the capabilities and limitations of sequential models in defect prediction tasks.

1.4 Objective

The aim of this review is to conduct a detailed analysis of the existing research on the use of sequential models for predicting software defects. Specifically, this review intends to categorize and identify the methodologies employed in using sequential models for defect prediction, including the model architectures, training strategies, and feature representations. The study also intends to scrutinize the datasets used in sequential model-based defect prediction studies, considering factors such as size, diversity, and representativeness. Furthermore, this review aims to evaluate the performance metrics and evaluation criteria used to assess the effectiveness of the sequential models in defect prediction tasks. Additionally, it aims to recognize the advantages, limitations, and practical considerations of using sequential models for defect prediction in real-world scenarios.

1.5 Thesis Structure

The thesis is structured to systematically and comprehensively address the objectives. Chapter 2, the Literature Review, provides an summary and overview of existing research on the application of sequential models for software defect prediction. It covers methodologies, datasets, evaluation metrics, and case studies. In Chapter 3, Methodology, the review's methodology is outlined, including search strategies, inclusion criteria, data extraction methods, and analysis techniques. Chapter 4, Findings and Analysis, presents the findings and provides a detailed analysis of the methodologies, datasets, evaluation metrics, and case studies identified in the literature. Finally, Chapter 5, Conclusion and Recommendations, this section highlights the key findings, examines their implications, and offers recommendations for directions in future research and practical applications.

This structure ensures a systematic and comprehensive examination of the application of sequential models for software defect prediction, facilitating a deeper understanding of their capabilities and limitations in real-world scenarios.

CHAPTER 2

LITERATURE REVIEW

2.1 Software Defect Prediction

An overview of recent studies conducted by different scholars on the topic of finding software flaws is given in this section. The research focuses on studies that used dimension reduction, pre-processing techniques, and sequential models to create a fault-free prediction model for software flaw prediction. Most research is used to build software prediction models, which enable software analysts to concentrate more on the code, give an estimate of the amount of work required, and determine whether the module has issues. This leads to the production of software that is of higher quality and makes better use of available resources.

Building prediction models to identify software flaws is the main task of SDP. While many methods and algorithms have been applied to enhance the performance of SDP models, Figure 1 provides a summary of the primary stages of SDP:

- Gather examples of both good and bad code from software sources.
- To build a dataset, extract features.
- If the dataset is unbalanced, correct it.
- Utilise the dataset to train a prediction model.
- Forecast the problematic components for a dataset taken from a fresh software project (or a modified version of the trained dataset).
- Assess the SDP model's performance.

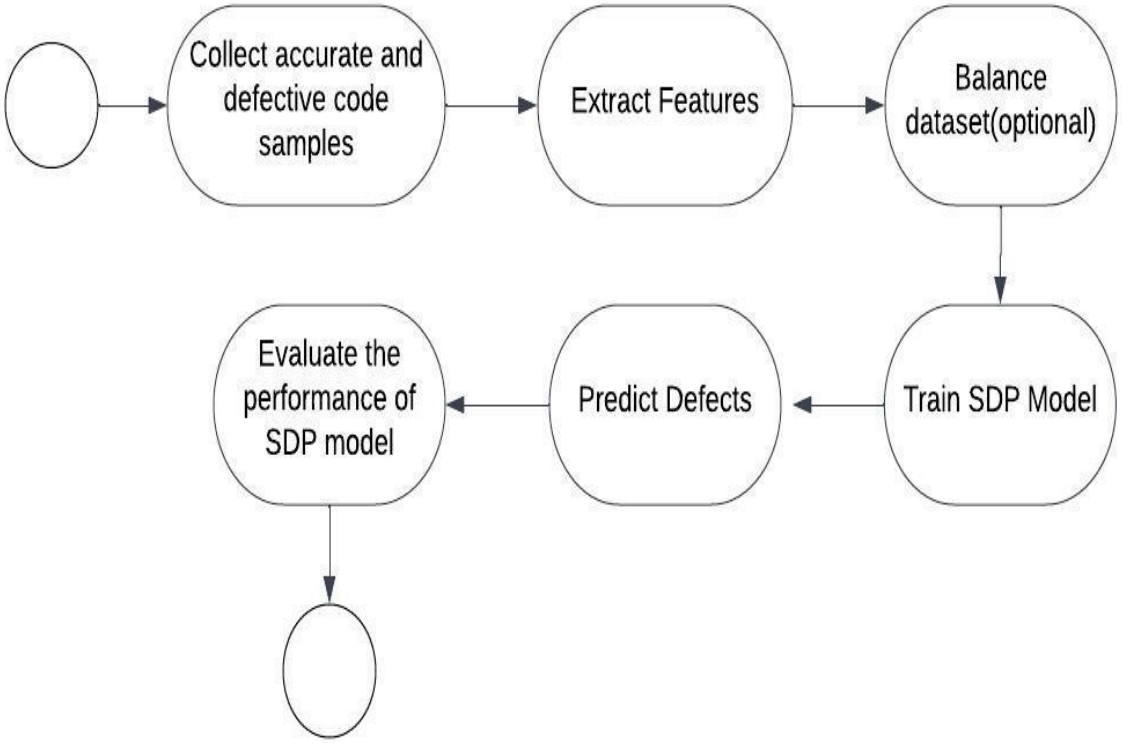

Figure 2.1: Software Defect Prediction Process

SDP process starts by collecting both flawless and flawed code samples from various sources such as source codes, commit messages, and bug reports. These samples are often obtained from archives and repositories. In the second phase of the process, metrics are extracted from the software artifacts such as source codes, commit logs, and messages. These metrics serve as important input data for model training. Feature extraction is a crucial step in the process as it involves dealing with different data types and representations. Data types range from McCabe metrics, CK metrics, change histories, assembly code, to source code. Besides traditional metric-based approaches, modern Deep Learning (DL) techniques can automatically extract features from high-dimensional and complex data.

An ensuing, optional step addresses data balance, imperative due to the typical scarcity of defective instances compared to non-faulty ones in defect datasets. Unfortunately, the class imbalance problem plagues most SDP techniques, impacting evaluation metrics. Techniques like oversampling are commonly employed to rectify this imbalance and enhance performance of SDP.

The step four in the process involves identifying any defective segments within the software. This requires a careful selection of deep learning (DL) algorithms and techniques, including various architectures like LSTM, and machine learning categories such as supervised or unsupervised. The level of granularity in identifying defects, whether it be at the module, file, class, function, or sentence level, is a critical consideration at this stage. Following this, the next step involves predicting defective segments within new test data by using the model trained in the previous stage. The predictions generated at this step serve as input for the final phase of the SDP process.

After the SDP process is completed, the final step involves evaluating the developed model. Evaluation metrics like F measure or AUC are used to systematically assess the predictive models. This facilitates a comparison with other relevant studies within the research domain.

2.2 Deep Learning based Sequential Models in SDP

Predicting software faults is an essential task in software engineering, which involves identifying potential defects in software systems before they arise. The use of SDP can assist software developers and quality assurance team in improving the software products quality and reducing maintenance cost. In recent years, sequential models have been employed to predict software defects, and the results have been promising.

Many research papers have shown that sequential models are effective in representing sequential data. These models have been utilized in defect prediction to simulate the temporal correlations between code changes and the emergence of defects in software systems. Sequential models can accept input sequences of varying lengths and can capture long-term dependencies in the data.

How accurately LSTMs forecast software faults has been the subject of numerous studies. Wang et al. [1] suggested a Gated Hierarchical LSTM-based approach to predict software system issues. In order to utilise both conventional and semantic data, A semantic level LSTM and a traditional level LSTM were combined to create a hierarchical LSTMs model. The PROMISE dataset, that is the most often used repository for software defect prediction research, is the source of the data that they choose to employ. They particularly used ten open source Java programmes from the repository. The semantic aspects are taught using the source code files' AST token sequences. They take the AST token sequences out of the source code files using the open source Python library javalang. The outcomes of GH-LSTM perform better.

Farid et al. [3] defined hybrid model CBIL for SDP. It is a combination of CNN and Bi-LSTM. CIBIL utilizes CNN to extract semantic features from source code files by using tokens generated by the AST whereas Bi-LSTM is used to save data of long term dependencies to extract necessary features. Nasir Uddin et al. [6] proposed the SDP-BB to predict vulnerabilities. Bi-LSTM, they were able to retrieve contextual information from the word embedding token vector. The CIBIL model is tested using seven open source Java apps from the Promise repository. They came to the conclusion that the CIBIL model outperforms the baseline model by 30% and, on average, improves CNN's F- score by 25%. For CPDP and WPDP, respectively, CBIL[3] and SDP-BB[6] are both useful. However, in terms of overall performance, CIBIL exceeds SDP-BB.

Munir et al. [2] presented a novel framework called DP-AGL that makes use of attention-based GRU-LSTM to forecast statement-level errors. They worked on the Bi-GRU and Bi-LSTM learning models as well as the 32 nodes for the metric. They took relevant properties from the LSTM output using an attention-based technique. DP-AGL was evaluated using 100,000 C/C++ programmes from the open source databank Code 4 Bench [2]. The source code was also parsed into an AST format using Clang. For recall, precision, accuracy, and F1 measures, the DP-AGL model performed on average at 0.98, 0.617, 0.75, and 0.757, respectively. In terms of performance, DP-AGL, a new method for sentence-level granularity, outperforms Random Forest and SLDeep [3].

A technique for prediction of statement-level software defects was developed by Majd et al. [4]. They employed LSTM as the learning model and 32 statement-level measures to analyse SLDeep. More than 100,000 C/C++ programmes from the open source repository Code 4 Bench were used to test this concept. To obtain accurate findings in this model, 10 fold cross validation was performed. Majd [4] claims that SLDeep could accurately anticipate phrases that were prone to errors with mean performance scores of 0.979, 0.570, and 0.702 in terms of recall, precision, and accuracy. Based on the aforementioned findings and validations, it can be said that SLDeep appears to be useful for predicting statement-level software defects.

CM_LSTM was developed by Pemmada et al. [5] to estimate software problems in software modules using modelled data. Target variables have been modified in light of the characteristics' positive connection with the target variable. In order to address the imbalance problem in the SDP data, the prepared data is supplied to the LSTM model. They used five fold cross validation to validate the model's ability to forecast software defects. The JM1 dataset,

which comprises 21 method-level attributes, was used for the experiment. To solve the problem of data imbalance, they employed feature selection technique. The accuracy of CM-LSTM [5] is 92.6%, outperforming C-LSTM and other machine learning methods, whose accuracy ranges from 75% to 81%.

A new framework for defect prediction called Seml was introduced by Liang et al. [8]. It combines deep learning and word embedding techniques. However, software measurements often miss the syntax and semantic content of programs. In order to solve this, a mapping table was created by transforming each token in the sequence into a real-valued vector using an unsupervised word embedding model. An LSTM network is then constructed using the vector sequences and their labels (defective or non-defective). The LSTM model is capable of predicting software defects and understanding the semantic content of programs. For the experiment, 13 projects were selected from the Apache Project repository and Github, which are available as open source. Seml [8] outperforms three cutting-edge defect prediction algorithms on most of the projects, giving better results for both WPDP and CPDP.

LSTM based on bidirectional and tree structure (LSTM-BT) was proposed by Zhot et al. [8]. Bidirectional LSTM (BI- LSTM) and tree LSTM are combined in the proposed model. They created this methodology to take syntactic and semantic information out of source code. From promise dataset eight java projects which are open source were taken to build this paradigm. An embedding layer is used to extract features from vectors taken from an abstract syntax tree (AST), which are then put into an LSTM-BT to generate predictions. This model beat a number of other defect prediction models in terms of the MCC and AUC metrics, according to our testing on 8 pairs of projects. Overall, our results demonstrate that LSTM-BT is an effective approach for improving SDP performance.

Hamza et al. [17] introduced a Layered Recurrent Neural Network (LRNN) coupled with wrapper feature selection algorithms (Binary Genetic Algorithm, Binary Particle Swarm Optimization, and Binary Ant Colony Optimization). The experiments, conducted on 19 Open Source Java Projects from the Promise Repository, reveal the LRNN's ability to achieve a noteworthy average classification rate of 0.8358 based on AUC results. The study underscores the importance of feature selection algorithms in Software Defect Prediction (SDP), positioning this iterative approach as a compelling method for enhancing software fault prediction across diverse Java projects.

Batool at al. [18] leverages two distinct datasets for analysis: Dataset-1, comprising 70 publicly available datasets with CK metrics (MFA, CA), and Dataset-2 sourced from the Git repository, consisting of 21 static metrics known as the GHPR dataset. Employing deep learning techniques, including LSTM, BI-LSTM, and RFCN, Batool conducts comprehensive experiments involving over 200 tests, varying hyperparameters, activation functions, and layer configurations. Notably, LSTM and BI-LSTM algorithms demonstrate superior performance with accuracies of 93.53% and 93.75%, respectively, while RFCN exhibits faster execution. The evaluation metrics encompass accuracy, recall, precision, F1-measure, and ROC-AUC, showcasing the outperformance of the proposed models against state-of-the-art approaches. However, the study acknowledges challenges in handling data imbalance, particularly in Dataset-1, where 82% of instances belong to the non-faulty class. The research employs 10-fold cross-validation, explores correlation coefficients, and adopts standardization for dataset normalization, contributing to a robust and meticulously conducted analysis in the domain of software fault prediction.

Wen et al. [24] and Chen et al. [29] both employ BI-LSTM architectures on ten Java Projects which are open source from the Promise dataset. Wen introduces BI-LSTM with a self-attention mechanism (BSLDP), showcasing its superiority over four baseline methods with significant improvements of 14.2%, 34.6%, 32.2%, and 23.6% in terms of F1 for defect prediction. The experiments highlighted that, in comparison to standard features examined by logistic regression, trained semantic features improve cross-project defect prediction by 34.5% in F1. However, the limitation lies in its application exclusively to Java-based projects. On the other hand, Chen employs BI-LSTM with attention in DeepCPDP, addressing the data imbalance problem through random under-sampling. The combined usage of SimASTToken2Vec, BiLSTM, and attention mechanisms proves highly effective, outperforming other state-of-the-art methods. The experiments, conducted within the CPDP framework, incorporate effort-aware evaluation measures like ACC and Popt, as well as non-effort aware measures such as AUC. Both studies contribute significantly to the domain of CPDP, demonstrating the efficacy of deep learning models in enhancing defect prediction across diverse projects.

A technique introduced by Tian et al. [28] combines the Gated Recurrent Unit (GRU) with the System Dependence Graph (SDG) in order to improve predictive performance. In the study, the method was evaluated on a set of ten Java Projects which are open source that were extracted from the Promise Repository. The results demonstrated a significant increase in predictive performance, with an average improvement of 11.0% in F1 measure for within-

project defect prediction and 10.4% in F1 measure for cross-project defect prediction when compared to the existing Tree-LSTM method. Precision, recall, and F-measure were all used as evaluation metrics. It should be noted that the experimentation was limited to Java projects and the study systematically applied SDG, a graphical representation that effectively captures semantic information and program dependence. Additionally, Word2Vec embedding was utilized to transform program slices into vectors, which bolstered the model's resilience.

Xing et al. [22] proposed a novel approach that combines GAN and BI-LSTM called G-LSTM, which has demonstrated superior performance in CPDP compared to conventional and state-of-the-art methods. The study collected data from five public projects in the Promise dataset, and the G-LSTM model outperformed other methods in terms of evaluation metrics like AUC and Accuracy. The model's generalizability is a significant strength, as it has been tested across 20 pairs of source-target projects. CBOW word embedding algorithm was used to develop this approach, which aligns with the CPDP framework.

Table 2.1 Summary of most recently used models and their parameters.

Author	**Year**	**Dataset**	**Accuracy Evaluation Parameters**	**Models**
Hao Wang, Weiyuan Zhuang, and Xiaofang Zhang [1]	2021	Promise	F-Measure, PofB20, IFA, Popt	LSTM
Hafiz Shahbaz Munir, Shengbing Ren Shazib Qayyum [2]	2021	Code4Bench	Recall, Precision, Accuracy, F1 Measure	LSTM with attention
Ahmed Farid, Ahmed Sharaf Eldin ,Enas Mohamed Fathy, and Laila A. Abd-Elmegid [3]	2021	Promise(7 Java Projects)	F-Measure and AUC	BI-LSTM and CNN
Amirabbas Majd, Alireza Khalilian, Mojtaba Vahidi-Asl, Pooria Poorsarvi Tehrani, Hassan Haghighi [4]	2019	Code4Bench	Precision, Recall, Accuracy, F-Measure	LSTM

Suresh Kumar Pemmada, Janmenjoy Nayak, H. S. Behera [5]	2022	JM1, Promise	Accuracy, F1-Score, ROC-AUC	LSTM
Md Nasir Uddin, Zafar Ali, Bixin Li, Pavlos Kefalas, Inayat Khan, Islam Zada[6]	2022	Promise	Precision, Recall, F1-Score	BI-LSTM
Xuan Zhou, Lu Lu [7]	2020	Promise	MCC, AUC	BI-LSTM and Tree-LSTM
Hongliang Liang, Yue Yu, Lin Jiang, Zhousi Xie [8]	2019	Promise(8 Java Projects), GitHub(5 Projects)	Precision, Recall, F1-Score	LSTM
Jiehan Deng, Lu Lu, Shaojian Qiu [9]	2020	Promise	F-Measure	BI-LSTM
Guisheng Fan, Xuyang Diao, Huiqun Yu, Kang Yang, Liqiong Chen [10]	2019	Apache	F1-Measure, AUC	RNN with attention
A J Anju, J. E. Judith [11]	2022	JM1	Accuracy, Precision, F1-score	RNN
Hoa Khanh Dam, Trang Pham [12]	2019	Samsung, Promise	Precision, AUC, Recall, F-Measure	Tree-LSTM
Guisheng Fan, Huiqun Yu, Kang Yang Xuyang Diao [13]	2019	Apache	F1-Measure	RNN with attention
Nasraldeen Alnor Adam Khleel, Karoly Nehez [14]	2023	Promise	Accuracy, AUC, Precision, Recall, F-Measure, MCC, AUCPR, MSE	GRU and CNN
Nasraldeen Alnor Adam Khleel [15]	2022	GHPR	Accuracy, AUC, Precision, Recall, F-Measure, MCC, MSE	CNN and Bi-LSTM

Prathyusha Tadapaneni, Naga Chandana Nadella, Mudili Divyanjali, Dr.Y.Sangeetha [16]	2022	NASA	Accuracy, MAE, MSE, R2 Score, RMSE	LSTM, Naïve Bayes, DNN
Hamza Turabieh, Majdi Mafarja, Xiaodong Li [17]	2018	Promise	AUC	RNN, Feature Selection Algorithms
Iqra Batool, Tamim Ahmed Khan [18]	2023	GHPR	Accuracy, Recall, Precision, F1-Measure, ROC-AUC	LSTM, BI-LSTM and RFCN
Nasraldeen Alnor Adam Khleel, Karoly Nehez [19]	2023	Promise, GitHub	Accuracy, F-Measure, Precision, MSE, Recall, MCC, AUC, AUCPR	LSTM and GRU
Sushant Kumar Pandey, Anil Kumar Tripathi [20]	2021	Promise	MSE, MAE, Accuracy	LSTM with attention
Emin Borandag [21]	2023	Eclipse, Jira	Accuracy, AUC	CNN, LSTM, BI-LSTM, RNN
Ying Xing [22]	2022	Promise(5 Java Projects)	AUC, Accuracy	GAN and BI-LSTM
Dingbang Fang , Shaoying Liu, Ai Liu [23]	2022	Promise(9 Java Projects)	Popt, F-Measure	BI-LSTM and ODCN
Wanzhi Wen [24]	2022	Promise(10 Java Projects)	F1-Measure	BI-LSTM with self attention.
Dingbang Fang, Shaoying Liu, Ai Liu [25]	2021	Promise(7 Java Projects)	F-Measure, AUC, MCC	BI-LSTM and ODCN
Xian Zhang, Kerong Ben, Jie Zeng [26]	2018	Promise	Precision, Recall, F1-Measure, AUC	LSTM
Junhao Lin, Lu Lu [27]	2021	Promise(8 Java Projects)	Recall, F-Measure, AUC, MCC	BI-LSTM
Junfeng Tian, Yongqing Tian [28]	2020	Promise(10 Java Projects)	Precision, Recall, F-Measure	GRU

Deyu Chen, Hao Li, Xiang Chen, Junfeng Xie, Yanzhou Mu [29]	2019	Promise(10 Java Projects)	ACC, Popt and AUC	BI-LSTM with attention
Hao Li, Xiang Chen, Xiaohong Li, Xiaofei Xie, Yanzhou Mu, Zhiyong Feng [30]	2019	Promise(10 Java Projects)	AUC	BI-LSTM
Hani Bani-Salameh, Bashar Al shboul, Mohammed Sallam [31]	2020	JIRA bug tracking system(5 closed-source projects)	Accuracy, Precision, Recall, F-Measure and MCC	RNN-LSTM

CHAPTER 3

RESEARCH OBJECTIVES AND METHODS

The methodology plays a crucial role to conduct this research, as it offers a methodical and organised strategy to organising, carrying out, deriving conclusions from, and disseminating research investigations. Researchers can ensure the validity, reliability, and generalizability of the research by following a rigorous methodology, as well as enhance its impact and relevance. The systematization of any research may be done through following steps:

1. Define the research question: Defining the research question that will be examined is the first stage review paper will address. Research questions are the central inquiries that guide a research study. They are the questions that the researcher seeks to answer through their investigation. A research question must be specific, answerable, clear, significant, feasible and relevant. A well formulated research question is essential for conducting a successful research study. Here are the steps to form research questions:

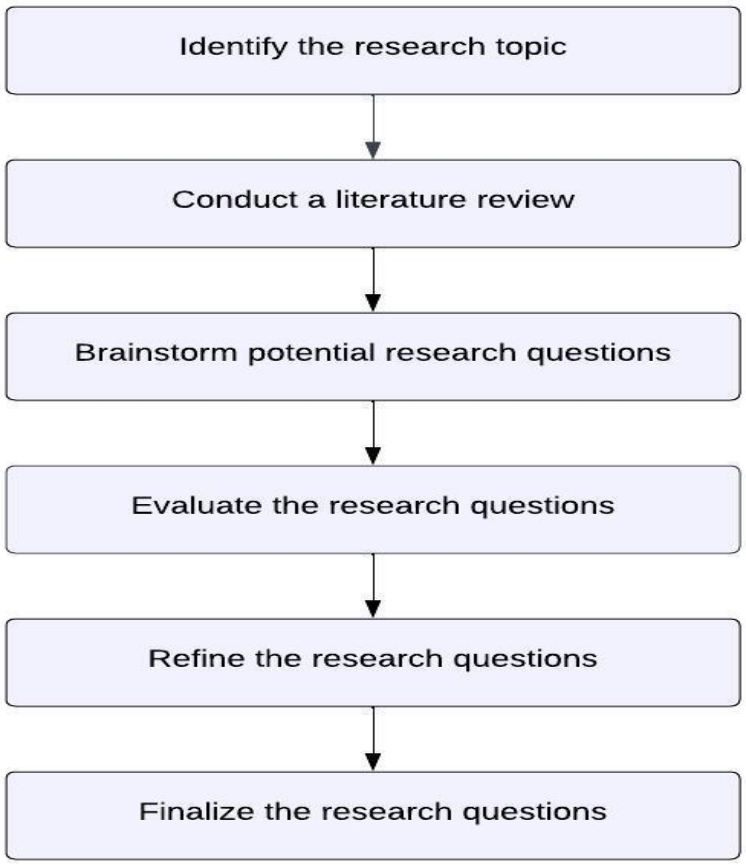

Figure 3.1: Steps to define the research question

- Identify the research topic: Finding the study subject is the first stage in developing research questions. The topic should be specific and focused. The topic chosen here is SDP using Convolutional Neural Network.
- Conduct a literature review: Forming research questions requires performing a literature study. It aids in determining the research and knowledge gaps that exist in the area.
- Brainstorm potential research questions: Based on the research topic and literature review, brainstorm potential research questions. These questions should be specific, clear, and answerable.
- Evaluate the research questions: Evaluate the potential research questions based on their relevance, feasibility, and significance. The research questions should be relevant to the research topic, feasible to answer, and significant in contributing to the existing knowledge.
- Refine the research questions: Refine the research questions based on the evaluation. The refined research questions should be specific, clear, answerable, relevant, feasible, and significant.
- Finalize the research questions: Finalize the research questions that will guide the research study.

The research question formulated are mentioned and described in the below table:

Table 3.1 Research questions and their description

ID	Research Question	Description
RQ1	What are the different sequential models employed for Software Defect Prediction?	This research question elaborate about the type of sequential models employed in the study of SDP using sequential models.
RQ2	What are the types of datasets used in the studies in SDP using sequential models?	This research question explains about which type of datasets are used in the studies in Software Defect Prediction(SDP) using sequential models whether Open Source, Small Experiments or Industrial .

RQ3	What are the most commonly used open-source datasets across studies in Software Defect Prediction(SDP) research?	This research question will provide further information about what are the most used datasets which are publicly available across studies.
RQ4	How many number of studies employed Effort-Aware and Non-Effort-Aware metrics for SDP using sequential models?	This research question will explore whether studies in SDP using sequential models employ effort-aware or non-effort-aware evaluation metrics.
RQ5	What are the most used metrics in non-effort aware scenarios across studies in SDP using sequential models?	In this research question what are the most commonly used metrics in non-effort-aware scenarios across studies will be discussed.
RQ6	What are the temporal distribution of studies over years in SDP using sequential models?	This research question describes about the interest of researchers or research attention in SDP using sequential models or number of studies across years published.

2. Conduct a literature search: In the next step perform a thorough literature search to identify all relevant studies and articles. This can be done using different academic databases such as MDPI, ACM Digital Library, Springer, Science Direct and IEEE Xplore. The search included keywords such as "Software defect prediction", "SDP", "Long short-term memory", “GRU” and "RNN". We have used the advanced search feature to input different combinations of keywords for searching the papers.

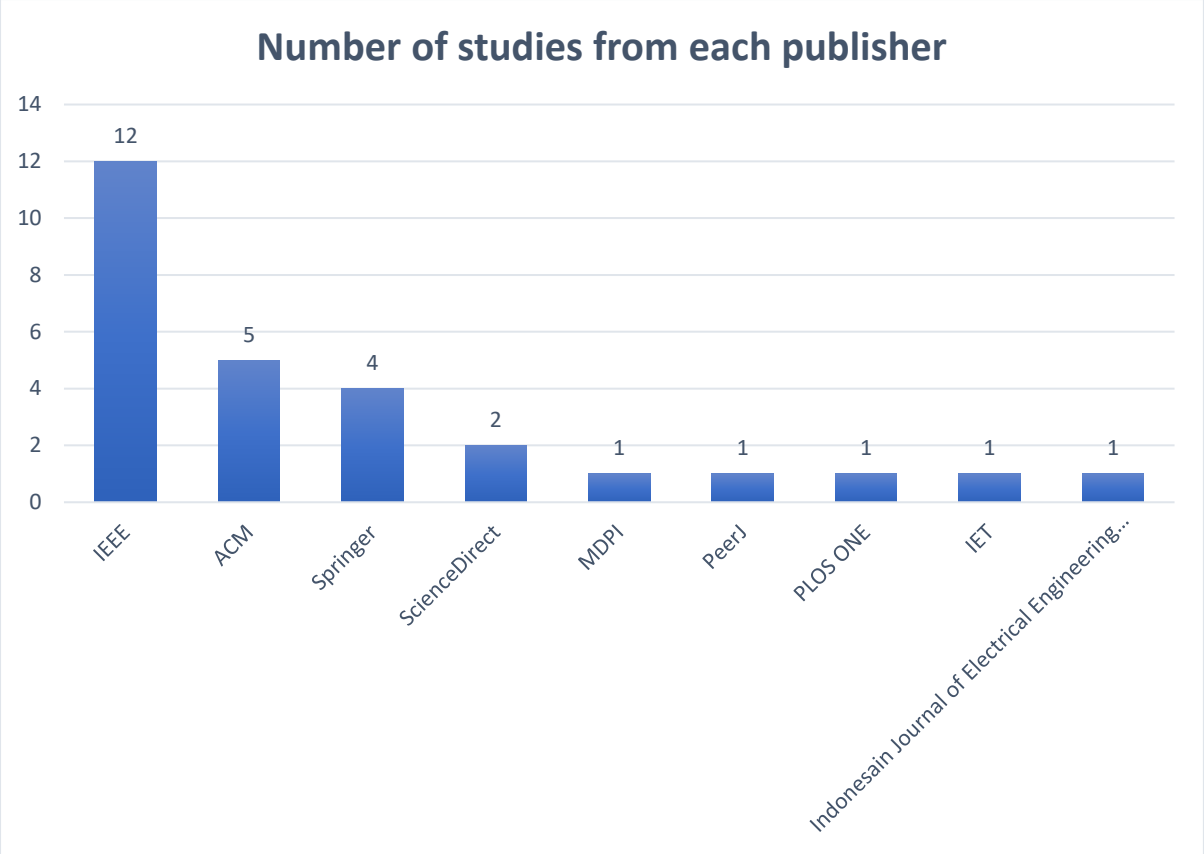

Figure 3.2: Number of studies that are reviewed from each publisher

3. Screen and select articles: After the literature search is finished, the following action is to screen and select articles that meet the inclusion criteria. The inclusion criteria used in the review include factors such as relevance to the research question, clarity of the technique used, well defined methodology and quality of the study.

4. Extract data: The following step is to extract information from the chosen relevant articles. This include information such as the type of sequential models used, the dataset used for training and testing, the performance metrics used, metrics used is whether effort-aware or non-effort aware, statistical test used, advantages and limitations of the study and the results obtained.

5. Analyze and synthesize data: Once the data has been extracted, the next step is to analyze and synthesize it. This could involve comparing and contrasting the results obtained by different studies, identifying common themes and trends, and highlighting any gaps or limitations in the existing research.

6. Write the review paper: Finally, the data analysis and synthesis can be used to write the review paper. The paper should include an introduction that provides background information

on SDP and various sequential models, literature review that summarizes the existing research, a discussion of the findings, along with a conclusion that summarizes the key findings and proposes areas for future research.

Figure 2 shows process of research which is presented below:

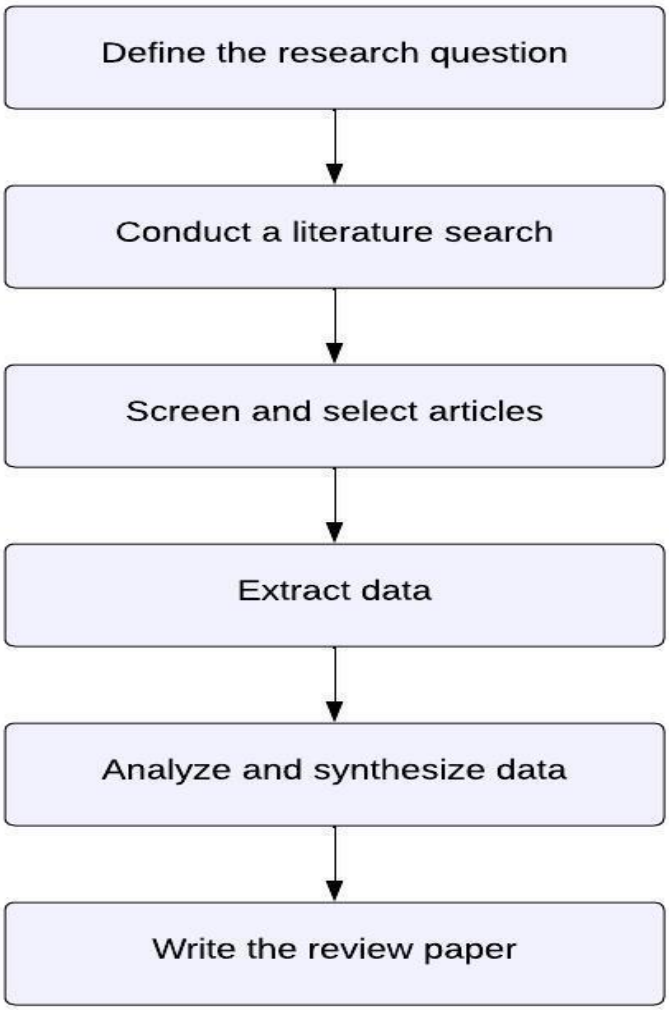

Figure 3.3: Process of Research

CHAPTER 4

RESULTS AND DISCUSSION

In this section, we present the results of the research on SDP utilizing sequential models. Our goal was to harness the power of models such as RNNs and LSTMs to enhance the accuracy of defect prediction in software development. Our experiments covered diverse datasets from varying software projects and domains, enabling us to evaluate the efficacy of these models. Our findings highlight the potential of sequential models to transform the field of software quality assurance by identifying and mitigating software defects. These findings contribute to the expanding knowledge base on advanced deep learning techniques for software engineering applications, underscoring the importance of sequential models in predicting software defects.

4.1 Commonly employed Sequential Models employed for Software Defect Prediction.

Deep learning techniques have brought a revolution in the field of SDP, offering advanced tools to analyze complex data from software development. Machine learning is used ingeniously through deep learning to automatically learn complex patterns, making predictive analytics a popular method in software engineering. Within this framework, sequential models are particularly significant, as they are capable of identifying and capturing the temporal dependencies that are inherent in the software development process.

Sequential models have several advantages over traditional approaches, especially in Software Defect Prediction. Unlike conventional methods that may struggle to capture intricate temporal relationships, sequential models excel in discerning nuanced patterns over time. This temporal awareness is crucial in software defect prediction, as the occurrence of defects often depends on the historical context of development activities. Moreover, sequential models can gather semantic features from the temporal evolution of software processes, providing an advantage over traditional, non-sequential models. This semantic understanding enhances the models' ability to discern subtle contextual nuances and relationships, resulting in more accurate and informed predictions of software defects.

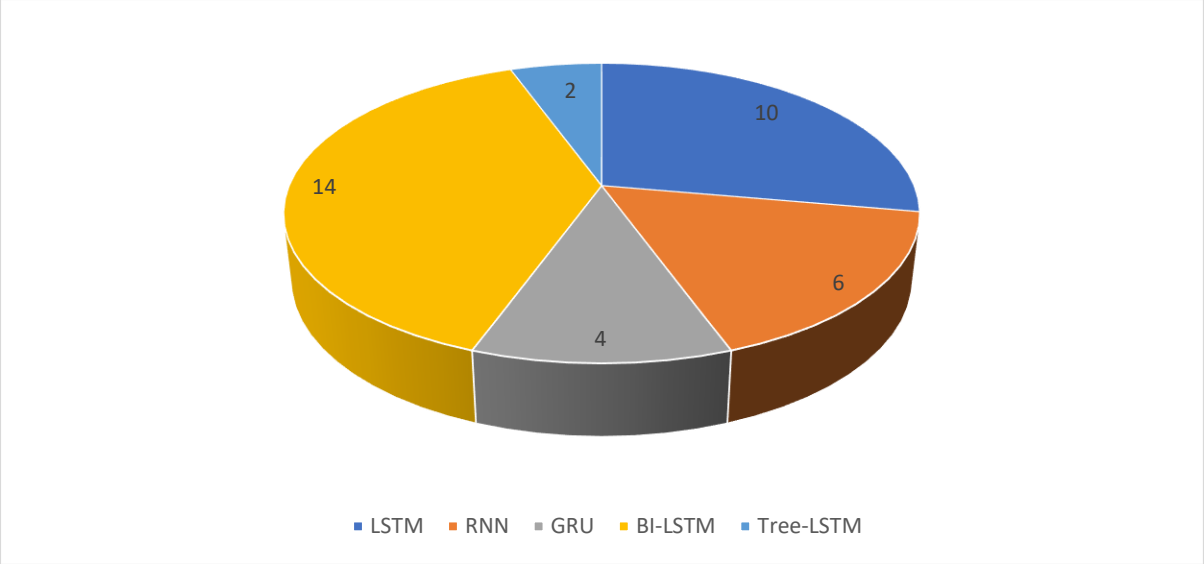

Figure 4.1: Number of studies for each sequential models employed.

In 10 studies, Long Short-Term Memory (LSTM)[1][8][20] was used due to its recurrent neural network (RNN) architecture that captures prolonged dependencies in sequential data. Recurrent Neural Network (RNN) was used in 6 studies, showing its versatility in handling sequential data for predictive analytics in detecting software defects. Additionally, Gated Recurrent Unit (GRU)[14][19][28] was applied in 4 studies, indicating its role as an alternative recurrent neural network architecture that has a simplified structure, yet remains effective in modeling sequential dependencies.

The use of Bidirectional LSTM (BI-LSTM)[3][6][9][15][22] is very common in software development, with 14 studies identifying it as a powerful model for processing input data in both directions to capture complex temporal dependencies. This makes BI-LSTM highly effective in predicting software defects. Additionally, Tree-LSTM[7][12] is mentioned in 2 studies as a specialized and innovative approach to sequential modeling for tree-structured data in the context of software defect prediction.

The frequent use of sequential models in software defect prediction research highlights their importance in advancing the field. These models are particularly effective in capturing and utilizing temporal dependencies within software development data. Bidirectional LSTM (BI-LSTM) is the most commonly used model, appearing in 14 studies, due to its proficiency in capturing nuanced temporal dependencies. Long Short-Term Memory (LSTM) is also widely used, appearing in 10 studies, which reinforces its relevance in the research landscape. The consistent use of advanced sequential models, including GRU, RNN[10][21][26], and Tree-

LSTM, confirms their significant impact in improving the methodology and reliability of software defect prediction models.

4.2 Types of datasets used in the studies in SDP using sequential models.

In the realm of SDP research that employs sequential models, the importance of datasets cannot be overstated. Datasets form the foundation on which predictive models are built and their selection plays a crucial role in ensuring the robustness and generalizability of the findings. It is imperative for researchers to carefully curate and explore diverse datasets in order to develop effective sequential models that can discern patterns and dependencies within software development processes.

Datasets play a crucial role in SDP research as they have the ability to capture the complexity and variability that is inherent in software development. These datasets are essentially repositories of historical information that offer valuable insights into the temporal evolution of projects. They also help in identifying factors that contribute to software defects. In the context of sequential models that are exceptional in capturing temporal dependencies, the importance of selecting the right datasets becomes even more critical. Given the complex and collaborative nature of software development, it is imperative to have datasets that emulate real-world scenarios.

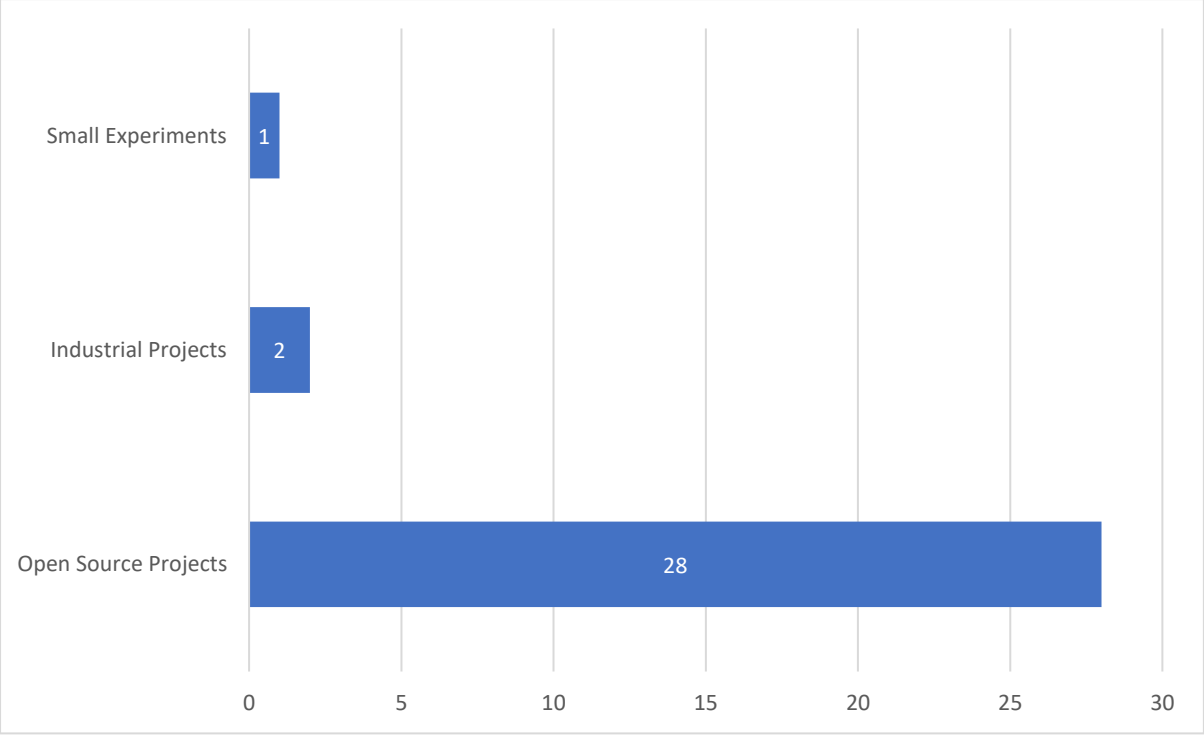

Figure 4.2: Different types of datasets used in studies

The types of datasets employed in SDP research using sequential models exhibit a spectrum of characteristics. Open Source Projects[1][3][4][11], often sourced from repositories like PROMISE, provide a wealth of publicly available data, reflecting the collaborative nature of open source software development. Industrial Projects[12][31], such as those from Samsung[12], introduce an element of real-world applicability, addressing challenges unique to commercial software development. Additionally, Small Experiments[21], involving datasets prepared by researchers, allow for controlled exploration and methodological advancements. This array of dataset types underscores the need for a multifaceted approach, considering the diverse contexts in which software defect prediction models are applied. In the subsequent analysis, we delve into the distribution and characteristics of these datasets in the context of SDP research utilizing sequential models.

4.3 Most commonly used open-source datasets across studies in Software Defect Prediction(SDP) research.

Several open-source datasets are commonly used in studies on SDP, including PROMISE[1][3][7][9][12], Code4Bench[2][4], JM1[5][11], Samsung[12], GHPR (GitHub)[15][18], and others. Among these, the PROMISE dataset is the most frequently used in 21 studies, indicating its prevalence and importance in the research landscape. Code4Bench and JM1, each utilized in two studies, contribute to the diversity of datasets employed in empirical research. The Samsung dataset, featured in one study, provides insights into software defect prediction challenges within proprietary software development environments. Additionally, the GHPR dataset[15][18] is utilized in two studies, demonstrating the increasing reliance on real-world, version-controlled data from collaborative platforms like GitHub. Apart from these explicitly mentioned datasets, three other datasets, collectively referred to as "Others," are employed in three studies, highlighting the continuous exploration of alternative datasets in the pursuit of comprehensive and robust software defect prediction models. The variety of datasets, encompassing different project types and metrics, underscores their collective significance in advancing the understanding of software quality and maintenance

effort in the software development process.

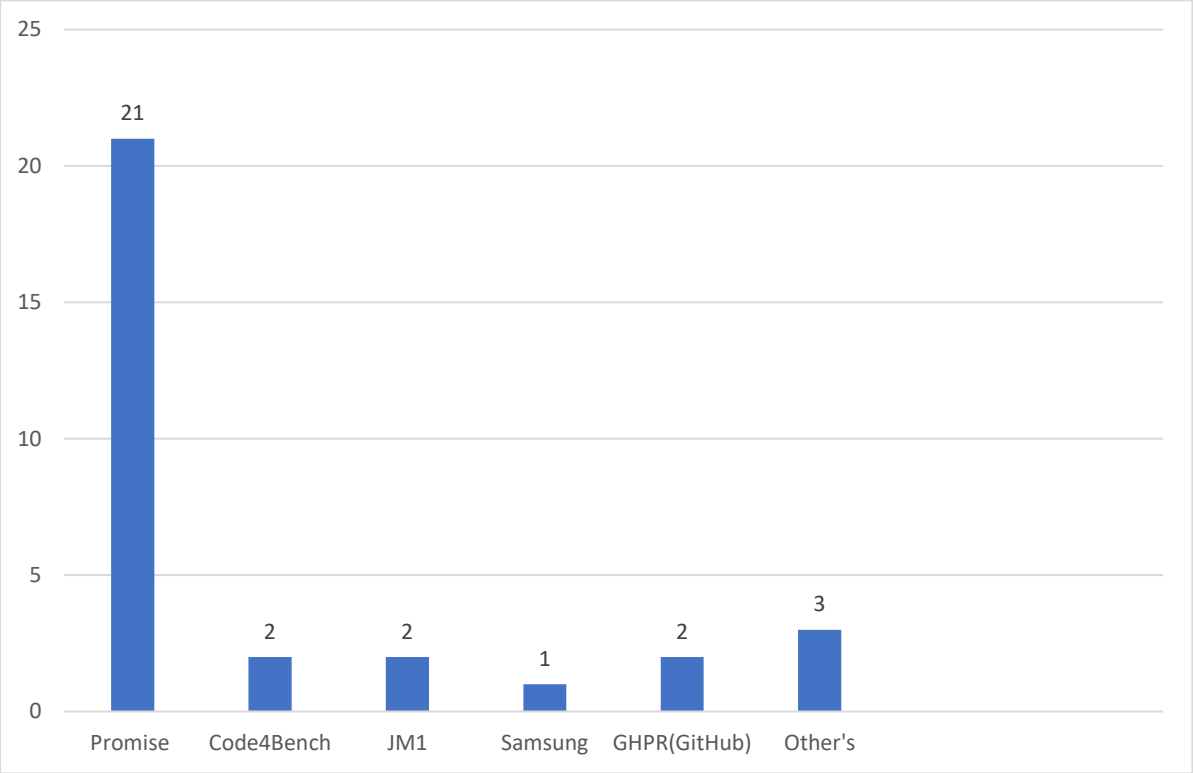

Figure 4.3: Common open source datasets across studies.

PROMISE (Predictive Models in Software Engineering): The PROMISE dataset [1][3][7][9][12] is a repository of software engineering datasets that includes data from various software projects, both open-source and proprietary. The repository contains datasets related to various aspects of software engineering, such as requirements engineering, effort estimation, and defect prediction. The goal of the PROMISE repository is to provide a standardized set of datasets for researchers to use in their studies, enabling more accurate comparisons between different studies and models. The Promise dataset consist of projects such as camel, xerxces, jEdit, xalan, ant, ivy, synapse, poi, etc.

Code4Bench:

The Code4Bench dataset[2][4] is designed as a comprehensive repository for software defect prediction, offering a diverse collection of C/C++ code contributions from different developers addressing varied programming problems. In contrast to the PROMISE dataset, Code4Bench is specifically tailored for the analysis of code quality and defect occurrences in C/C++ projects. Code4Bench offers an extensive dataset for researchers to delve into the complexities of software defect prediction within the C/C++ programming domain. The structured format of the dataset, including tables explicitly specifying defective and correct versions, enhances

its usability for studies focused on understanding the intricate relationships between code features and defect occurrences in this specific programming context.

In Software Defect Prediction (SDP) research, utilizing various datasets with their specific strengths and contexts enriches collective knowledge. Open-source datasets such as Promise, Code4Bench, JM1, Samsung, GHPR (GitHub), and others, including NASA[16] and JIRA[21] datasets, offer a wide range of nuances. NASA dataset[16] is recognized for its broad application across various software engineering domains and provides a valuable resource for analyzing software defects. On the other hand, JIRA[21], a widely used issue tracking system, offers insights into different issue types, including bug reports and enhancements. This inclusive approach to dataset utilization deepens researchers' insights into the multifaceted nature of software defect prediction when employing sequential models. It encompasses both traditional code-centric repositories and broader issue-tracking perspectives. The collective exploration of these datasets contributes to advancing the methodologies and reliability of software defect prediction models, fostering continuous improvement in the understanding and mitigation of software defects.

4.4 Number of studies employed Effort-Aware and Non-Effort-Aware metrics for SDP using sequential models.

In the field of Software Defect Prediction the distinction between Effort-Aware[1][23][29] and Non-Effort-Aware metrics[2][4][5][6][8][11][13][23] takes on a crucial role in shaping the methodologies employed by researchers. Effort-Aware metrics[1][23][29], such as Popt, PofB20, IFA, and ACC, focus on capturing factors related to the effort expended during the software development process. These metrics often include measures that assess the resource allocation, complexity, and time spent in creating and maintaining software. Integrating Effort-Aware metrics into SDP models acknowledges the intrinsic connection between the effort exerted in development and the likelihood of encountering defects. This approach enables researchers to gain insights not only into the probability of defects but also into the resource implications associated with different software components.

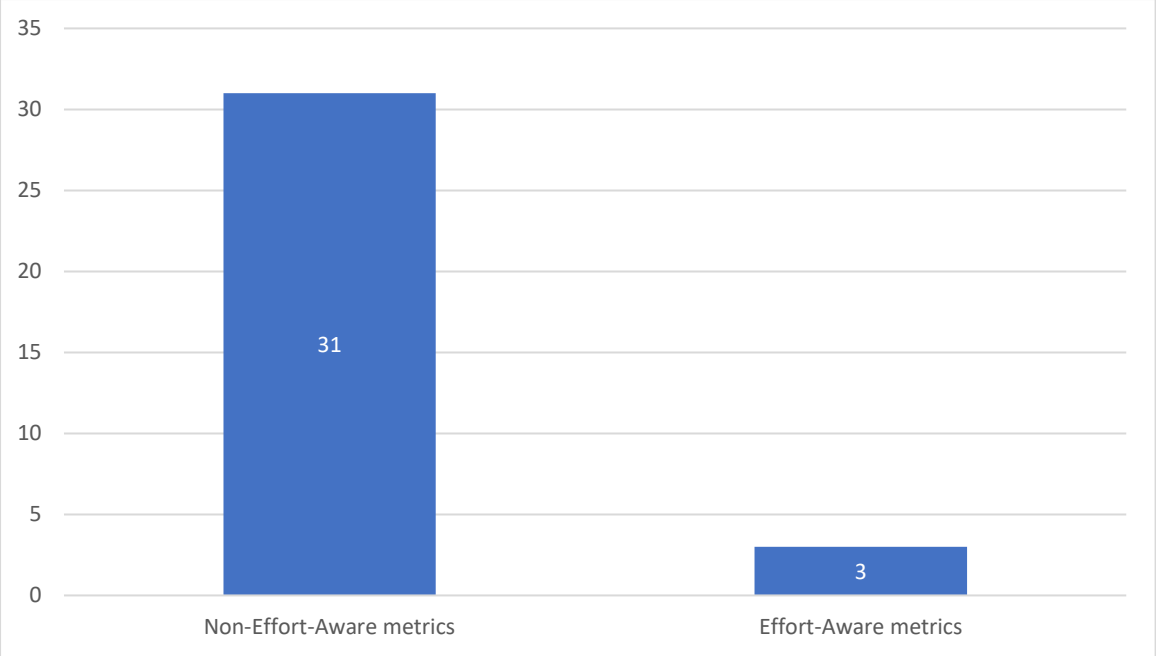

Figure 4.4: Number of studies that used effort-aware and non-effort-aware metrics

On the other hand, there are Non-Effort-Aware metrics in SDP[2][4][5][6][8][11][13][23] that focus exclusively on intrinsic code characteristics such as precision, recall, and accuracy, rather than the effort or resources invested in software development. These metrics, which include F-Measure, Precision, Recall, Accuracy, and F1-Measure, are useful in identifying defect-prone areas and providing a more code-centric perspective. Non-Effort-Aware metrics are important because they allow us to assess the inherent properties of the code and predict software defects without considering development effort.

In the landscape of SDP studies employing sequential models, the distribution of the usage of Effort-Aware and Non-Effort-Aware metrics is noteworthy. Thirty one studies have opted for Non-Effort-Aware metrics, emphasizing the significance of code-centric characteristics in predicting software defects without explicit consideration of development effort. In contrast, a more limited number of studies, precisely three, have integrated Effort-Aware metrics, reflecting a subset of research endeavors that seek to understand the interplay between development effort and defect occurrences. This diversity in metric selection highlights the nuanced approaches within SDP research, allowing for a comprehensive exploration of factors contributing to software defects when utilizing sequential models.

4.5 Metrics used in non-effort-aware scenarios across studies in SDP using sequential models.

In the domain of Software Defect Prediction using sequential models, the selection and evaluation of appropriate metrics are paramount to assess the predictive efficacy of models without explicit consideration of development effort. Non-Effort-Aware metrics play a pivotal role in this context, focusing exclusively on intrinsic code characteristics and defect occurrences, disregarding the resources or effort invested in the software development process. These metrics provide a code-centric perspective, allowing researchers to gain insights into the inherent properties of the codebase and its predictive capabilities. Various non-effort-aware metrics that were used in studies are:

1. **F-Measure:** F-Measure is a widely employed metric in Non-Effort-Aware scenarios, with 13 studies incorporating it into their analyses. F-Measure combines precision and recall, offering a unbiased assessment of a models's results in predicting both positives and negatives[1][3][9].

2. **Precision:** Precision is a key metric used in 14 studies, highlighting its importance in evaluating the accuracy of positive predictions made by a model. It is particularly relevant in scenarios where minimizing false positives is crucial[2][4][18].

3. **Recall:** Recall, used in 15 studies, focuses on the ability of a model to identify all relevant instances of a positive class. It is important in situations where capturing as many true positives as possible is a primary concern[6][8][12].

4. **Accuracy:** Accuracy, featured in 13 studies, provides a holistic measure of a model's correctness by considering both true positives and true negatives in relation to the total instances. It is a fundamental metric in classification scenarios[16][18][31].

5. **F1-Measure:** F1-Measure, which appears in 11 studies, evaluates a performance of model by considering both false positives and false negatives, offering a balanced assessment of precision and recall[10][12][28].

6. **AUC:** Area Under the Receiver Operating Characteristic Curve (AUC), which is utilized in 17 studies, assesses the ability of the model to differentiate between positive and negative instances by varying the thresholds. It is widely adopted as a metric for binary classification tasks [3][5][18][22][25].

7. **MCC (Matthews Correlation Coefficient):** MCC, featured in 7 studies, takes into consideration true and false positives and negatives, providing a balanced measure that is particularly robust for imbalanced datasets[7][14][25].

8. **AUCPR (Area Under the Precision-Recall Curve):** AUCPR, employed in 2 studies, evaluates the model's performance by considering precision and recall across different decision thresholds, offering insights into classification performance in imbalanced datasets[14][19].

9. **MSE (Mean Squared Error):** Five studies used Mean Squared Error to quantify prediction accuracy, measuring the mean squared difference between actual and predicted values[14][19][20].

10. **MAE (Mean Absolute Error):** Mean Absolute Error, appearing in 2 studies, evaluates the average absolute difference between predicted and actual values, offering insights into the magnitude of errors[16][20].

11. **R2 Score:** R2 Score, as featured in one study, measures the proportion of variance in the dependent variable that can be predicted by the independent variables. It provides an indicator of prediction quality[16].

12. **RMSE (Root Mean Squared Error):** Root Mean Squared Error, utilized in 1 study, is a variation of Mean Squared Error, offering a measure of the average magnitude of the errors[16].

13. **KE (Kappa Error):** Kappa Error, featured in 1 study, assesses the agreement between predicted and observed values while considering the agreement that could occur by chance[1].

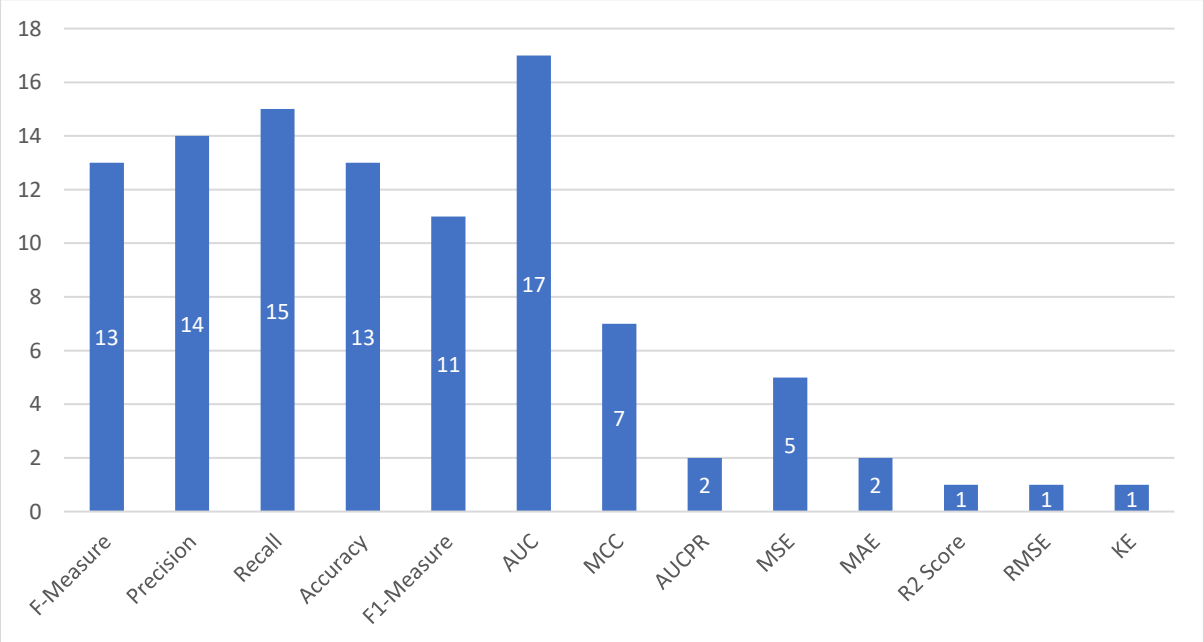

Figure 4.5: Most used metrics in non-effort-aware scenarios

Among these metrics, AUC[3][5][18][22][25] stands out as the most commonly utilized, featured in 17 studies, showcasing its widespread adoption for assessing model performance in distinguishing between positive and negative instances. In contrast, R2 Score[16], Kappa Error (KE)[1], and Root Mean Squared Error (RMSE)[16] are the least employed, each appearing in only one study. The limited use of these metrics could be attributed to their specific applicability or relevance in certain specialized contexts. While AUC serves as a robust indicator of classification performance, the less frequently used metrics like R2 Score, KE, and RMSE might cater to unique requirements within the broader SDP research landscape. This distribution highlights the importance of tailored metric selection based on the specific goals and characteristics of individual studies.

4.6 Distribution of studies over years in SDP using sequential models.

The use of sequential models, such as LSTM [1][8][20], GRU [14][19][28], and Bidirectional LSTM (BI-LSTM) [3][6][9][15][22] in Software Defect Prediction (SDP) has marked a significant shift in recent years. In 2018, a seminal paper[17][26] was published introducing the use of a sequential model for SDP, which has since gained considerable traction. This pioneering work has paved the way for subsequent studies to explore the potential of sequential models in predicting software defects. The increasing number of studies from 2018 onwards underscores the growing interest and recognition of the effectiveness of sequential models in the field of SDP. Researchers are now relying more on these models to improve predictive accuracy and reliability in identifying possible software defects.

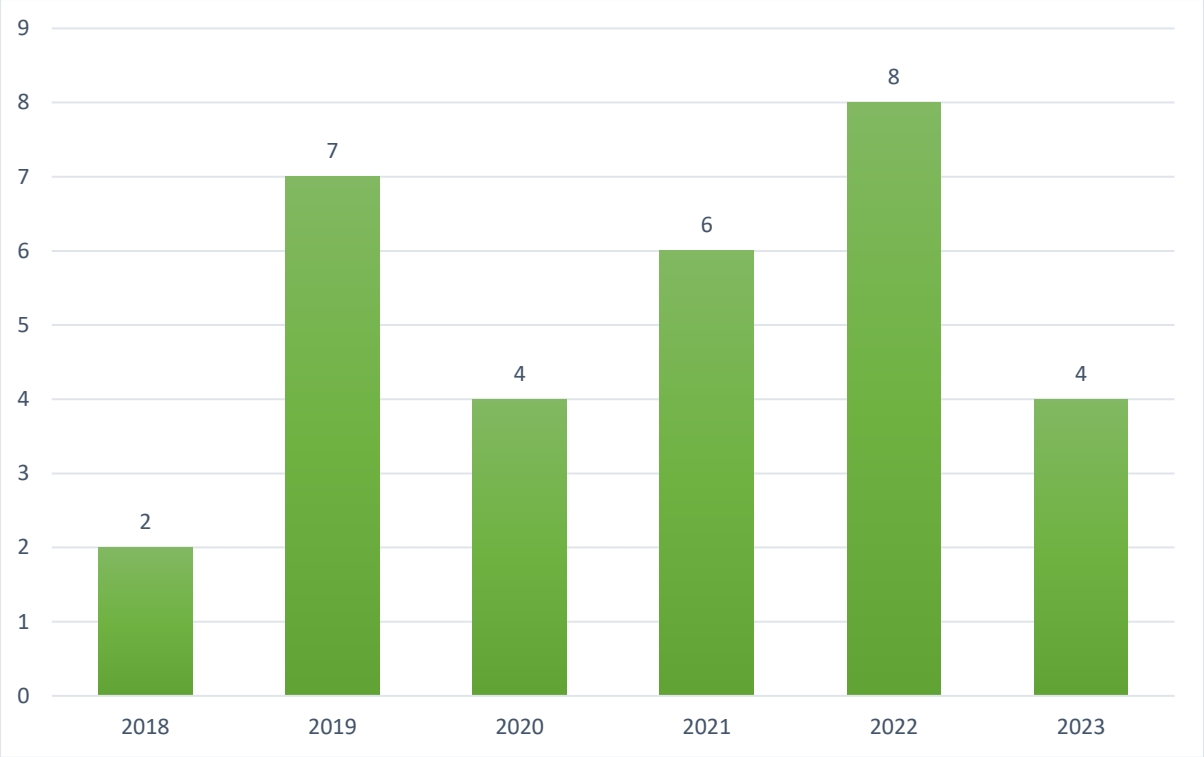

Figure 4.6: The distribution of studies over years

The use of sequential models like LSTM, GRU, BI-LSTM has become common in Software Defect Prediction (SDP) research. This is because these models are capable of capturing semantic features and temporal dependencies within software development data. Unlike traditional deep learning models, sequential models are proficient in recognizing patterns and trends over time. This is highly beneficial in software development as it is inherently dynamic. Sequential models can retain and process information across sequential data, which enables a nuanced understanding of the temporal relationships between different code components. This, in turn, contributes to an enhancement in defect prediction accuracy. Therefore, researchers can more effectively identify and mitigate potential software defects.

Software defect prediction (SDP) research has seen a significant rise in the use of sequential models lately. The year 2022 witnessed a surge in research activity with the publication of eight papers utilizing sequential models, indicating a growing interest and appreciation for their effectiveness in SDP. Researchers are exploring and leveraging LSTM, GRU, BI-LSTM, and other models to advance the methodologies and reliability of SDP beyond traditional deep learning models. With the distinct advantages of sequential models and the increasing interest in them, they are expected to become pivotal tools in the ongoing evolution of SDP methodologies.

CHAPTER 5

CONCLUSION AND FUTURE WORK

The application of sequential models, such as Long Short-Term Memory (LSTM) [1][8][20], Bi-Directional LSTM (Bi-LSTM) [3][6][9][15][22], and Gated Recurrent Unit (GRU) [14][19][28], in software defect prediction has been a significant development in recent years. Among these models, Bi-Directional Long Short-Term Memory (Bi-LSTM) is the most commonly used in software defect prediction. However, the reliance on the Promise repository as the primary dataset highlights the need for diversification in data sources to improve the applicability of predictive models to real-world software environments. It is worth noting that most studies mainly use non-effort aware metrics, leaving a gap in the integration of effort-aware metrics[1][23][29], like Popt and PofB20 in defect prediction. This observation calls for a more comprehensive and nuanced approach to evaluating software defect prediction models that aligns with the multifaceted nature of software development and quality assessment.

In order to improve the accuracy and practical applicability of predictive models for software defect analysis, it is important to prioritize the exploration of diverse datasets obtained from real-world software systems. Currently most of the researchers use dataset from promise repository[1][3][7][9][12], so, reliance on a single dataset may introduce biases and limit the representativeness of research findings. Collaborative efforts with industrial partners can facilitate the acquisition of such diverse datasets, contributing to the robustness of predictive models. By expanding the scope of datasets, researchers can gain a more comprehensive understanding of software defects, which will help to improve the reliability of software systems in the future.

In addition, the positive results achieved by hybrid models, such as the combination of Bi-LSTM and GRU[19], suggest that they should be given more attention in future research. Hybrid models demonstrate the possibility of combining different sequential architectures to improve predictive performance. Analyzing the complementary strengths of various sequential models and hybrid approaches can lead to the development of more robust and versatile software defect prediction models. It is essential to conduct further research on improving advanced sequential models by diversifying datasets, exploring hybrid models, and addressing the challenges associated with interpretability. The transparency of model decision-making

processes should be enhanced, especially in safety-critical applications, to instill confidence in the predictions made by these models.

To sum up, the research findings stress the significance of diversifying datasets for software defect prediction, exploring hybrid models, and tackling interpretability challenges. By following these avenues, the research community can enhance the state-of-the-art in software defect prediction, leading to better software quality and decreased development costs.

REFERENCES

[1] Wang, H., Zhuang, W., & Zhang, X. (2021). Software defect prediction based on gated hierarchical LSTMs. *IEEE Transactions on Reliability*, *70*(2), 711-727.

[2] Munir, H. S., Ren, S., Mustafa, M., Siddique, C. N., & Qayyum, S. (2021). Attention based GRU-LSTM for software defect prediction. *Plos one*, *16*(3), e0247444.

[3] Farid AB, Fathy EM, Sharaf Eldin A, Abd-Elmegid LA. 2021. Software defect prediction using hybrid model (CBIL) of convolutional neural network (CNN) and bidirectional long short-term memory (Bi-LSTM) *PeerJ Computer Science* 7:e739 https://doi.org/10.7717/peerj-cs.739

[4] Majd, A., Vahidi-Asl, M., Khalilian, A., Poorsarvi-Tehrani, P., & Haghighi, H. (2020). SLDeep: Statement-level software defect prediction using deep-learning model on static code features. *Expert Systems with Applications*, *147*, 113156.

[5] Pemmada, S.K., Behera, H.S., Nayak, J. *et al.* Correlation-based modified long short-term memory network approach for software defect prediction. *Evolving Systems* **13**, 869–887 (2022). https://doi.org/10.1007/s12530-022-09423-7.

[6] Uddin, M.N., Li, B., Ali, Z. *et al.* Software defect prediction employing BiLSTM and BERT-based semantic feature. *Soft Comput* **26**, 7877–7891 (2022). https://doi.org/10.1007/s00500-022-06830-5.

[7] X. Zhou and L. Lu, "Defect Prediction via LSTM Based on Sequence and Tree Structure," 2020 IEEE 20th International Conference on Software Quality, Reliability and Security (QRS), Macau, China, 2020, pp. 366-373, doi: 10.1109/QRS51102.2020.00055.

[8] H. Liang, Y. Yu, L. Jiang and Z. Xie, "Seml: A Semantic LSTM Model for Software Defect Prediction," in IEEE Access, vol. 7, pp. 83812-83824, 2019, doi: 10.1109/ACCESS.2019.2925313.

[9] Deng, J., Lu, L. and Qiu, S., 2020. Software defect prediction via LSTM. *IET software*, *14*(4), pp.443-450.

[10] Fan, G., Diao, X., Yu, H., Yang, K. and Chen, L., 2019. Software defect prediction via attention-based recurrent neural network. *Scientific Programming*, *2019*.

[11] Anju, A.J., Judith, J.E. Adaptive recurrent neural network for software defect prediction with the aid of quantum theory- particle swarm optimization. *Multimed Tools Appl* **82**, 16257–16278 (2023). https://doi.org/10.1007/s11042-022-14065-7

[12] H. K. Dam et al., "Lessons Learned from Using a Deep Tree-Based Model for Software Defect Prediction in Practice," 2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR), Montreal, QC, Canada, 2019, pp. 46-57, doi: 10.1109/MSR.2019.00017.

[13] G. Fan, X. Diao, H. Yu, K. Yang and L. Chen, "Deep Semantic Feature Learning with Embedded Static Metrics for Software Defect Prediction," 2019 26th Asia-Pacific Software Engineering Conference (APSEC), Putrajaya, Malaysia, 2019, pp. 244-251, doi: 10.1109/APSEC48747.2019.00041.

[14] Khleel, N.A.A., Nehéz, K. A novel approach for software defect prediction using CNN and GRU based on SMOTE Tomek method. *J Intell Inf Syst* **60**, 673–707 (2023). https://doi.org/10.1007/s10844-023-00793-1.

[15] Nehéz, K. and Khleel, N.A.A., 2022. A new approach to software defect prediction based on convolutional neural network and bidirectional long short-term memory. *Production Systems and Information Engineering*, *10*(3), pp.1-18.

[16] P. Tadapaneni, N. C. Nadella, M. Divyanjali and Y. Sangeetha, "Software Defect Prediction based on Machine Learning and Deep Learning," 2022 International Conference on Inventive Computation Technologies (ICICT), Nepal, 2022, pp. 116-122, doi: 10.1109/ICICT54344.2022.9850643.

[17] Turabieh, H., Mafarja, M. and Li, X., 2019. Iterated feature selection algorithms with layered recurrent neural network for software fault prediction. *Expert systems with applications*, *122*, pp.27-42.

[18] Batool, I., Khan, T.A. Software fault prediction using deep learning techniques. *Software Qual J* **31**, 1241–1280 (2023). https://doi.org/10.1007/s11219-023-09642-4.

[19] Khleel, Nasraldeen & Nehéz, Károly. (2023). Improving the accuracy of recurrent neural networks models in predicting software bug based on undersampling methods. Indonesian Journal of Electrical Engineering and Computer Science. 32. 478-493.

10.11591/ijeecs.v32.i1.pp478-493.

[20] Pandey, S.K. and Tripathi, A.K., 2021. DNNAttention: A deep neural network and attention based architecture for cross project defect number prediction. *Knowledge-Based Systems*, *233*, p.107541.

[21] E. Borandag, “Software Fault Prediction Using an RNN-Based Deep Learning Approach and Ensemble Machine Learning Techniques,” *Applied Sciences*, vol. 13, no. 3, p. 1639, Jan. 2023, doi: 10.3390/app13031639. Available: http://dx.doi.org/10.3390/app13031639.

[22] Xing, Y., Qian, X., Guan, Y., Yang, B. and Zhang, Y., 2022. Cross-project defect prediction based on G-LSTM model. *Pattern Recognition Letters*, *160*, pp.50-57.

[23] D. Fang, S. Liu and A. Liu, "Gated Homogeneous Fusion Networks With Jointed Feature Extraction for Defect Prediction," in IEEE Transactions on Reliability, vol. 71, no. 2, pp. 512-526, June 2022, doi: 10.1109/TR.2022.3165115.

[24] W. Wen et al., "A Cross-Project Defect Prediction Model Based on Deep Learning With Self-Attention," in IEEE Access, vol. 10, pp. 110385-110401, 2022, doi: 10.1109/ACCESS.2022.3214536.

[25] D. Fang, S. Liu and A. Liu, "EPR: a Neural Network for Automatic Feature Learning from Code for Defect Prediction," 2021 IEEE 21st International Conference on Software Quality, Reliability and Security (QRS), Hainan, China, 2021, pp. 482-492, doi: 10.1109/QRS54544.2021.00059.

[26] X. Zhang, K. Ben and J. Zeng, "Cross-Entropy: A New Metric for Software Defect Prediction," 2018 IEEE International Conference on Software Quality, Reliability and Security (QRS), Lisbon, Portugal, 2018, pp. 111-122, doi: 10.1109/QRS.2018.00025.

[27] J. Lin and L. Lu, "Semantic Feature Learning via Dual Sequences for Defect Prediction," in IEEE Access, vol. 9, pp. 13112-13124, 2021, doi: 10.1109/ACCESS.2021.3051957.

[28] J. Tian and Y. Tian, "A Model Based on Program Slice and Deep Learning for Software Defect Prediction," 2020 29th International Conference on Computer Communications and Networks (ICCCN), Honolulu, HI, USA, 2020, pp. 1-6, doi: 10.1109/ICCCN49398.2020.9209658.

[29] D. Chen, X. Chen, H. Li, J. Xie and Y. Mu, "DeepCPDP: Deep Learning Based Cross-Project Defect Prediction," in IEEE Access, vol. 7, pp. 184832-184848, 2019, doi: 10.1109/ACCESS.2019.2961129.

[30] H. Li, X. Li, X. Chen, X. Xie, Y. Mu and Z. Feng, "Cross-project Defect Prediction via ASTToken2Vec and BLSTM-based Neural Network," 2019 International Joint Conference on Neural Networks (IJCNN), Budapest, Hungary, 2019, pp. 1-8, doi: 10.1109/IJCNN.2019.8852135.

[31] Bani-Salameh, H., Sallam, M. and Al shboul, B., 2021. A deep-learning-based bug priority prediction using RNN-LSTM neural networks. *e-Informatica Software Engineering Journal*, *15*(1).