Review

A Systematic Literature Review of Software Defect Prediction Using Deep Learning

[1,2]Ahmed Bahaa, [1]Enas Mohamed Fathy, [1,3]Ahmed Sharaf Eldin and [1]Laila A. Abd-Elmegid

[1]*Department of Information Systems, Faculty of Computers and Artificial Intelligence, Helwan University, Helwan 11795, Egypt*
[2]*Department of Information Systems, Faculty of Computers and Artificial Intelligence, Beni-Suef University, Beni-Suef 62521, Egypt*
[3]*Department of Information Systems, Faculty of Information Technology and Computer Science, Sinai University, Sinai, Egypt*

Article history
Received: 26-02-2021
Revised: 18-04-2021
Accepted: 11-05-2021

Corresponding Author:
Enas Mohamed Fathy
Department of Information Systems, Faculty of Computers and Artificial Intelligence, Helwan University, Helwan 11795, Egypt
Email: enasm.fathy@gmail.com

Abstract: The approaches associated with software defect prediction are used to reduce the time and cost of discovering software defects in source code and to improve the software quality in the organizations. There are two approaches to reveal the software defects in the source code. The first approach is concentrated on the traditional features such as lines of code, code complexity, etc. However, these features fail to extract the semantics of the source code. The second one is concentrated on revealing these semantics. This paper presents a Systematic Literature Review (SLR) of software defect prediction using deep learning models. This SLR is focused on identifying the studies that use the semantics of the source code for improving defect prediction. This SLR aims to analyze the used datasets, models and frameworks. Also, identifying the evaluation metrics to ensure their applicability in software defect prediction. IEEE Xplore, Scopus and Web of Science digital libraries were used to select the suitable primary studies. Forty (40) primary studies were selected that published by 15 December 2020 for analysis based on the quality criteria. The project levels that applied in the studies were: Within-project 52.5%, cross-project 17.5% and both within-project and cross-project 30%. The datasets used were: Promise dataset 68.18% and other datasets 31.82%. The most used deep learning model in the primary studies was: Convolutional Neural Network (CNN) by 35%. The most used evaluation metrics were: F-measure and Area Under the Curve (AUC). Software defect prediction using deep learning models is still a valuable topic and requires much research studies to enhance the performance of the defect prediction.

Keywords: Systematic Literature Review, Software Defect Prediction, Deep Learning, Semantics, Abstract Syntax Tree

Introduction

Proactive software testing plays a crucial role in the software development life cycle to find more defects earlier (Olsen, 2019). If the defects are discovered earlier through the prediction process, then the quality of software will be enhanced effectively. So, software defect prediction is becoming a popular research area in the field of software engineering.

Software defect prediction (Lin *et al*., 2018) is a process of predicting the software defects that occur in the source code by using historical information such as code complexity. It consists of four phases: The first phase is to collect and label the data to defective and clean files. There are different datasets for the experiments such as PROMISE and NASA repositories. The second step is to collect the key features of these files. The third phase is to build and train the model. In order to evaluate the performance of the proposed model, evaluation metrics can be used such as F-measure, AUC, etc. Then, the classifier will predict if the new data is defective or clean (Yang *et al*., 2015).

Software defect prediction is classified into Within-Project Defect Prediction (WPDP) and Cross-Project Defect Prediction (CPDP) (Tong *et al*., 2018). In WPDP, the collected data are retrieved from the releases in the same project. In CPDP, the collected data are retrieved from different projects.

There are some studies that have been conducted in software defect prediction. Wahono (2015) presents a systematic literature review of software defect prediction. The main goal of this study is to identify and analyze the datasets, methods and frameworks that used in software defect prediction between 2000 and 2013. This study analyzes the studies in software defect prediction based on statistical and machine learning models. However, this study does not include the evaluation metrics to evaluate the performance of the models. Also, it does not cover deep leaning models. Hosseini *et al*. (2017) also present an SLR to identify the metrics, models, data approaches, datasets and associated performances for CPDP. The results show that CPDP can achieve a good performance compared to WPDP, when enhancing the factors that impact the performance of the models.

Unlike these research studies, this SLR concentrates on identifying, analyzing and evaluating the datasets, models, frameworks and evaluation metrics used in software defect prediction using deep learning models. Therefore, it can be a good guide to obtain the suitable datasets, models and evaluation metrics that can be used in the future experiments.

The rest of the paper is organized as follows: Section II describes the SLR methodology. Section III presents the results of applying the quality criteria to the selected primary studies. Section IV presents the discussion of this SLR. Section V presents and summarizes the conclusion of this SLR.

Methodology

In order to conduct a deep analysis of software defect prediction, a Systematic Literature Review (SLR) was selected in this research. SLR collects the data from selected research studies to systematically deduce the results. SLR evaluates all the empirical research evidence to answer specific research questions (Torres-Carrión *et al*., 2018). It uses explicit criteria for deciding which studies will be included or excluded. This helps to minimize the authors' bias. SLR process consists of three phases (Okoli, 2015): Planning, conducting and reporting the literature review. In the planning phase, a review protocol is developed. It defines the research questions, search strategy, inclusion and exclusion criteria for selecting relevant studies. Quality assessment, data extraction and data synthesis are applied to the selected primary studies in the conducting phase. The results are presented in the reporting phase of the literature review. SLR steps are shown in Fig. 1.

A. Phase One: Planning the Literature Review

In the review protocol, the first step is to identify the purpose of the research. Then, research questions are formed to support the objective of the SLR. Then, the data are extracted based on the identified inclusion and exclusion criteria.

Research Questions

The objective of this SLR is to summarize, analyze and assess the empirical evidence regarding the datasets, models, frameworks and evaluation metrics used in the context of WPDP and CPDP. In WPDP, the training set and test set are retrieved from the same project. In CPDP, the training set and test set are retrieved from different projects, where the model is trained by a training set from one project and the test set is validated from another project. We define six Research Questions (RQs) to achieve the goal of this SLR. The research questions are developed based on the Population, Intervention, Comparison, Outcomes and Context (PICOC) (Sabir *et al*., 2019) criteria. Table 1 shows the PICOC criteria of this SLR. Then, the research questions are developed and evaluated based on the PICOC criteria. Table 2 shows the research questions and the description of each question.

Table 1: PICOC Criteria

Population	Datasets, Software projects, open-source projects, Software applications.
Intervention	Software defect prediction, software fault prediction, datasets, models, frameworks, evaluation metrics.
Comparison	Not Available.
Outcomes	Performance of software defect prediction models.
Context	Public and private datasets.

Table 2: Research Questions

ID	Research question	Description
RQ1	Which kind of project level is the most used for software defect prediction?	Identify the kind of project-level, either WPDP or CPDP.
RQ2	What is the type of datasets that is the most used for software defect prediction?	Identify the datasets commonly used in software defect prediction.
RQ3	What are the models of deep learning that are used for software defect prediction?	Identify the most used models in software defect prediction.
RQ4	What are the evaluation metrics that are used for software defect prediction?	Identify the evaluation metrics used in software defect prediction.
RQ5	What are the frameworks used in software defect prediction?	Identify the frameworks used in software defect prediction.
RQ6	What is the number of publications in software defect prediction over the years?	Identify the number of publications in software defect prediction each year.

Search Strategy

Search strategy, including search string and relevant repositories. The aim of the search strategy is to find all relevant studies that support the research questions. We apply the following steps to develop the search string. First, essential search terms are identified. Second, the synonyms of search terms are identified. Third, a combination of logical operators ANDs and ORs are used in the search process.

The following search string is used:

(("Deep Learning" OR "Recurrent Neural Network" OR "Convolutional Neural Network" OR "Deep Belief Network" OR "Long Short-Term Memory" OR "Autoencoder" OR "Deep Reinforcement Learning" OR "Generative Adversarial Network") AND ("Abstract Syntax Tree" OR "AST*" OR "Semantic*") AND ("Software" AND ("Defect" OR "Fault") AND "Prediction"))*

Choosing suitable repositories is considered a vital point of SLR to increase the retrieval of highly relevant studies. The most popular repositories are considered relevant in software engineering and have the opportunity to deal with complex search queries. The selected digital libraries are:

- IEEE Xplore
- Scopus
- Web of Science

The studies are retrieved by using title, abstract and keyword. The result of the studies in all digital libraries are merged, then the duplicated studies are removed. We select all the studies that were published by 15 December 2020. The studies are limited to publications of journals and conferences.

Inclusion and Exclusion Criteria

According to SLR, we should define a set of rules for choosing the most relevant studies. The inclusion and exclusion criteria are applied for selecting the primary studies. Table 3 shows the details of the inclusion and exclusion criteria.

B. Phase Two: Conducting the Literature Review

Select Primary Studies

Figure 2 shows the overview of identified and remained studies after each step in the search process. In total, 90 primary studies are retrieved from the selected repositories based on the defined search string. After removing the duplicated studies, 44 have remained. Then the irrelevant studies are discarded based on reading the details of each study; four studies are discarded, leaving 40 primary studies to be evaluated in this SLR.

Quality Assessment

The quality assessment ensures the efficiency of the studies and the eligibility for applying software defect prediction models. The quality assessment is focused on extracting the studies that have sufficient information for answering the predefined research questions. We have defined a set of quality criteria to be applied to the selected primary studies. The quality criteria are shown in Table 4. Each quality criteria must be answered using the options "Yes" or "No". The answer "Yes" represents value 1 and "No" represents value 0. We accumulate the values of all answers for each primary study. The sum is shown in Table 5. The primary study that reaches a sum lower than 80% will be excluded. After applying quality assessment on 40 primary studies, all 40 studies are included for the data extraction.

Data Extraction

The purpose of data extraction is to extract the data from the primary studies to answer the predefined research questions. The data extraction consists of three steps: The first step is concentrated on general information about the studies such as authors, publication title, publication type and publication year. The second step is focused on the implementation of software defect prediction models such as kinds of projects, datasets, models, frameworks and evaluation metrics. The third step extracts information about the empirical study and the final results of the models. Table 6 shows the characteristics that are used to answer the research questions. Table 7 shows the relationship between the primary studies and research questions, it checks if the selected studies answer the research questions or not.

Table 3: Inclusion and Exclusion Criteria

Inclusion criteria	
✓	Studies present an empirical study.
✓	Studies compare the performance of models.
✓	Studies concentrate on predicting defects that exist in a specific area of source code.
✓	Studies that published either in journals or conferences.
Exclusion criteria	
☒	Studies are not related to deep learning models.
☒	Studies are not discussing semantics and syntactics of source code.
☒	Studies are not written in English.

Table 4: Quality criteria

ID	Quality criteria	Description
Q1	Is the source of the used datasets clearly reported?	The source of datasets must be stated.
Q2	Is the prediction model trained and tested on a different type of data?	The model must be trained by the training set and tested by the test set.
Q3	Are the static code metrics clearly reported?	It is essential that static code metrics must be explicitly identified, such as lines of code, code complexity, etc.
Q4	Is the deep learning model clearly reported?	The deep learning model (e.g., Deep Belief Network, Recurrent Neural Network, etc.) must be clearly stated in the study.
Q5	Is the model applied to both within-project and cross-project?	The performance of the model must be reported in within-project and cross-project.
Q6	Are the evaluation metrics used clearly mentioned?	The metrics (e.g., F-measure, AUC, etc.) that evaluate the model must be reported.
Q7	Are the predictive values of the evaluation metrics clearly reported?	The predictive values of the evaluation metrics must be clearly represented in numbers.

Table 5: Results of quality criteria for the primary studies

Study	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Sum
St1 (Deng *et al*., 2020a)	1	1	1	1	0	1	1	6
St2 (Cai *et al*., 2019)	1	1	1	1	0	1	1	6
St3 (Sheng *et al*., 2020)	1	1	1	1	0	1	1	6
St4 (Meilong *et al*., 2020)	1	1	1	1	0	1	1	6
St5 (Humphreys and Dam, 2019)	1	1	1	1	0	1	1	6
St6 (Pan *et al*., 2019)	1	1	1	1	0	1	1	6
St7 (Phan *et al*., 2018a)	1	1	1	1	0	1	1	6
St8 (Wang *et al*., 2016)	1	1	1	1	1	1	1	7
St9 (Nivetha and Kavitha, 2019)	1	1	1	1	0	1	1	6
St10 (Phan and Le Nguyen, 2017)	1	1	1	1	0	1	1	6
St11 (Phan *et al*., 2017)	1	1	1	1	0	1	1	6
St12 (Li *et al*., 2019)	1	1	1	1	0	1	1	6
St13 (Qiu *et al*., 2019a)	1	1	1	1	0	1	1	6
St14 (Wang *et al*., 2018)	1	1	1	1	1	1	1	7
St15 (Fan *et al*., 2019a)	1	1	1	1	0	1	1	6
St16 (Chen *et al*., 2019)	1	1	1	1	0	1	1	6
St17 (Zhou *et al*., 2019)	1	1	1	1	0	1	1	6
St18 (Phan *et al*., 2018b)	1	1	1	1	0	1	1	6
St19 (Wen *et al*., 2018)	1	1	1	1	0	1	1	6
St20 (Huo *et al*., 2018)	1	1	1	1	1	1	1	7
St21 (Dam *et al*., 2019)	1	1	1	1	1	1	0	6
St22 (Shi *et al*., 2020)	1	1	1	1	1	1	1	7
St23 (Liang *et al*., 2019)	1	1	1	1	1	1	1	7
St24 (Fan *et al*., 2019b)	1	1	1	1	0	1	1	6
St25 (Li *et al*., 2017)	1	1	1	1	0	1	1	6
St26 (Deng *et al*., 2020b)	1	1	1	1	0	1	1	6
St27 (Zhang and Wu, 2020)	1	1	1	1	0	1	1	6
St28 (Tian and Tian, 2020)	1	1	1	1	1	1	1	7
St29 (Zhu *et al*., 2020a)	1	1	1	1	1	1	1	7
St30 (Fan *et al*., 2018)	1	1	1	1	0	1	1	6
St31 (Zhang *et al*., 2018a)	1	1	1	1	0	1	1	6
St32 (Qiu *et al*., 2019b)	1	1	1	1	0	1	1	6
St33 (Wang and Lu, 2020)	1	1	1	1	1	1	1	7
St34 (Chen *et al*., 2020)	1	1	1	1	1	1	1	7
St35 (Zhu *et al*., 2020b)	1	1	1	1	0	1	1	6
St36 (Zhang *et al*., 2020)	1	1	1	1	0	1	1	6
St37 (Dong *et al*., 2018)	1	1	1	1	1	1	1	7
St38 (Zhou and Lu, 2020)	1	1	1	1	0	1	1	6
St39 (Shi *et al*., 2021)	1	1	1	1	1	1	1	7
St40 (Hoang *et al*., 2020)	1	1	1	1	0	1	1	6

Table 6: Data extraction characteristics mapped to research questions

Characteristic	Research question
Authors, publication title, type, year	General
Project levels	RQ1
Datasets	RQ2
Deep learning models	RQ3
Evaluation metrics	RQ4
Frameworks	RQ5
Progress of publications	RQ6

Table 7: The relationship between the primary studies and research questions

Study	RQ1	RQ2	RQ3	RQ4	RQ5	RQ6
St1	√	√	√	√	√	√
St2	√	√	√	√	√	√
St3	√	√	√	√	√	√
St4	√	√	√	√	√	√
St5	√	√	√	√	√	√
St6	√	√	√	√	√	√
St7	√	√	√	√	√	√
St8	√	√	√	√	√	√
St9	√	√	√	√	√	√
St10	√	√	√	√	√	√
St11	√	√	√	√	√	√
St12	√	√	√	√	√	√
St13	√	√	√	√	√	√
St14	√	√	√	√	√	√
St15	√	√	√	√	√	√
St16	√	√	√	√	√	√
St17	√	√	√	√	√	√
St18	√	√	√	√	√	√
St19	√	√	√	√	√	√
St20	√	√	√	√	√	√
St21	√	√	√	√	√	√
St22	√	√	√	√	√	√
St23	√	√	√	√	√	√
St24	√	√	√	√	√	√
St25	√	√	√	√	√	√
St26	√	√	√	√	√	√
St27	√	√	√	√	√	√
St28	√	√	√	√	√	√
St29	√	√	√	√	√	√
St30	√	√	√	√	√	√
St31	√	√	√	√	√	√
St32	√	√	√	√	√	√
St33	√	√	√	√	√	√
St34	√	√	√	√	√	√
St35	√	√	√	√	√	√
St36	√	√	√	√	√	√
St37	√	√	√	√	√	√
St38	√	√	√	√	√	√
St39	√	√	√	√	√	√
St40	√	√	√	√	√	√

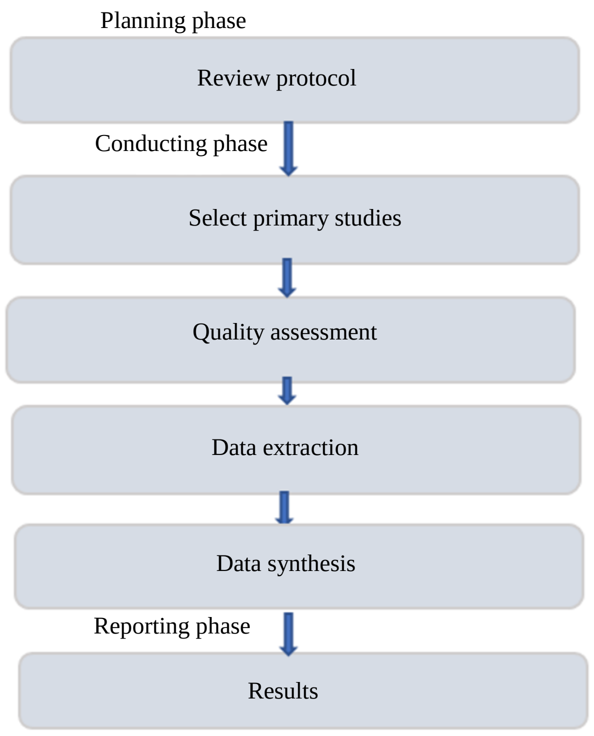

Fig. 1: SLR steps

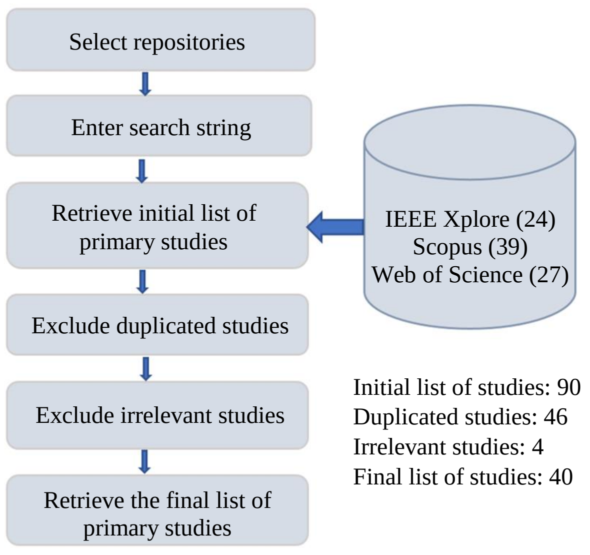
Select repositories
Enter search string
Retrieve initial list of primary studies
Exclude duplicated studies
Exclude irrelevant studies
Retrieve the final list of primary studies
IEEE Xplore (24)
Scopus (39)
Web of Science (27)
Initial list of studies: 90
Duplicated studies: 46
Irrelevant studies: 4
Final list of studies: 40

Fig. 2: Selection of relevant studies

Data Synthesis

The goal of data synthesis is to collect the data together from the selected primary studies to answer the research questions, with the aim of aggregating the evidence (Huang *et al*., 2018). There are many types of strategies to synthesize the data. Meta-analysis is used when the studies are homogeneous. Therefore, it cannot be applied in this SLR because the primary studies discuss different deep learning models, datasets, frameworks and evaluation metrics. So, we use the narrative synthesis strategy. In narrative synthesis, visualization of the results is presented with the support of adding text to explain the context. In this SLR, the data are presented in a manner compatible with the research questions. The results of the primary studies are presented by using tables, bar charts, pie charts and column charts. The results are shown in section III.

Phase Three: Results

A. Project Levels

Project level has two kinds: Within-project and cross-project. Within-project means both of the training set and test set are retrieved from the same project (i.e., Project A). While cross-project means training set and test set are retrieved from different projects (i.e., Project A, Project B) where the model is trained by training set from Project A and test set is retrieved from Project B.

In Within-Project Defect Prediction (WPDP), there are two versions (pre, post) of the same project. Pre version is used for training set and post version is used for test set. In Cross-Project Defect Prediction (CPDP), there are two different projects (source project, target project). Source project is used for training set and target project is used for test set.

Figure 3 shows the kind of project-level, either within-project or cross-project, that implemented in the selected primary studies. Within project was applied to 21 primary studies, while 12 studies used both of within-project and cross-project and only 7 studies applied cross-project.

For example, St8 used PROMISE dataset. It applied both of WPDP and CPDP. For WPDP, Ant (1.5 and 1.6) were selected. Ant 1.5 was chosen as training set and Ant 1.6 was chosen as test set. For CPDP, Ant 1.6 and Camel 1.4 were selected. Ant 1.6 as training set and Camel 1.4 as test set. For all the Forty primary studies, the percentage of each kind of project-level is shown in Fig. 4. Within-project was applied in most of the primary studies. However, some models achieved good performance in within-project and bad results with cross-project. To check the performance of the model, it should be applied for within-project and cross-project.

B. Datasets

Dataset is a data collection that is used to train and test machine learning and deep learning models. In this SLR, 40 primary studies are analyzed to find which dataset is used in them. Table 8 shows the details of the datasets used for each study. After that, for each dataset, we calculate the number of the primary studies that applied it. There are public and private datasets that used in software defect prediction. There is no standard dataset to be used in the models, because each organization has its own dataset. Therefore, the public datasets are the best choice for researchers to build their models. However, public datasets may have some issues such as lack of data quality. For example, there are two versions of PROMISE (Ferenc *et al*., 2018) dataset. Simplified PROMISE Source Code (SPSC) and PROMISE Source Code (PSC). Pan *et al*. (2019) worked on PROMISE dataset and built the SPSC. The difference in the datasets will affect the evaluation of the proposed models.

As shown in Fig. 5, most of the studies use one dataset and few studies use two datasets. And also, PROMISE dataset is mostly used in software defect prediction. PROMISE was used in 30 primary studies. It is a public repository. It contains the following projects: Ant, Camel, Forrest, Ivy, Jedit, Log4j, Lucene, Pbeans, Poi, Synapse, Velocity, Xalan and Xerces. It is used for many purposes. For example: Jedit is a text editor designed for programmers to help them. Poi is a Java library used to access Microsoft format files. And Xalan is a library used to transform XML documents. NASA dataset was used in two primary studies. It is a public repository. In St35 and St36, KC2, MC1, PC1 and PC2 projects were selected from NASA dataset. AEEM dataset was used in only one primary study, it includes Eclipse and Apache. The other studies used the following datasets: Open-source projects, GitHub projects, Codeforces projects, CodeChef projects android projects, C# projects and OJ system projects. All the selected datasets in the primary studies are public datasets, because they are available for all researchers. Figure 6 shows the percentage of the PROMISE dataset and other datasets that are used in the primary studies.

To apply the datasets in the proposed models, researchers select the datasets, then the projects are chosen. The datasets contain the traditional features and defect data of source files in each project. Each project includes project name, project versions, average number of source files and average defect rate. The first step is to label the data to be defective or clean for each file. The second step is to collect the traditional features of these files. The third step is to build the model. Finally, trained models are used to predict if the new instance is defective or clean.

C. Models

Recently, deep learning has emerged as a powerful model to improve the effectiveness of software defect prediction. Deep Learning (DL) models are Deep Neural Network (DNN), Deep Belief Network (DBN), Convolutional Neural Network (CNN), Recurrent Neural Network (RNN), Bidirectional Recurrent Neural Network (Bi-RNN), Long Short-Term Memory (LSTM), Bidirectional Long Short-Term Memory (Bi-LSTM), Attention Mechanism (AM) and Autoencoder (AE). DL models performs better results for software defect prediction. In DL models, huge data are used to train the models. This data will enhance the performance of the proposed models. Moreover, DL models can extract the semantics of the source code effectively. And this helps to predict the software defects before the execution of the software. And also, DL models reduce the time and cost of detecting the software defects in the source code. In this SLR, there are 40 primary studies that use deep learning models. Table 9 shows which deep learning model is used in each study. Figure 7 shows the number of the studies for each DL model. The percentage of each model is shown in Fig. 8.

DNN (Samir *et al*., 2019) and DBN (Yuan *et al*., 2020) outperform machine learning models. They use a neural network with multi-levels. The levels consist of an input layer, many hidden layers and an output layer. Semantic features are extracted from source code. Then the extracted features will be used to build the model. CNN (Dahou *et al*., 2019) achieves better performance than DNN and DBN. CNN uses convolutions and pooling in order to produce feature maps and reduce the dimensionality of the output. This will help to extract semantic features of Abstract Syntax Tree (AST) tokens more effectively. Recently, RNN (Staudemeyer *et al*., 2019) and its types achieve good results over the years. RNN is a sequential model. It uses a neural network with internal memory to track the states of every input in the network. RNN passes information through the forward direction, while Bi-RNN passes information in both forward and backward directions. LSTM is a special type of RNN. It contains a cell for memory to track the information of long-term dependencies. Bi-LSTM consists of two independent LSTM. It passes the information through forward and backward directions. Attention Mechanism (Li and Liu, 2018) is a sequential model. It concentrates on the importance of each node of the sequence by determining the weight of each node. It sets a high weight for the important data to enhance defect prediction. Attention Mechanism is used with other DL models such as Bi-LSTM and RNN. It achieves better results when it is used alone and also when it combines with Bi-LSTM or RNN. The recent deep learning model is the Autoencoder (Zhu *et al*., 2019). It copies the data from the input to the output. It encodes the input values then decodes the encoded values. It extracts the most robust features by minimizing the reconstruction error between the input and output.

After analyzing the primary studies, we find that the most used DL model in the primary studies is CNN. CNN extracts semantics of source code more effectively. It can capture the best feature representation of source code. In addition, it captures more defect feature information from source code. Moreover, the time and accuracy of the used classifiers will be enhanced. CNN has the ability to decrease the classification error between source and target projects. Furthermore, it generates more semantic features when code comments are embedded.

D. Evaluation Metrics

Evaluation metrics are used to evaluate the performance of machine learning and deep learning models. Main metrics (Li *et al*., 2018; Tantithamthavorn *et al*., 2018) are Precision, Recall, F-measure and Area Under the Curve (AUC). We compared the performance of the primary studies based on F-measure and/or AUC because most of the studies applied them. When comparing the proposed models with the other selected baseline models, most of the studies applied one evaluation metric (F-measure or AUC) and few studies applied both metrics (F-measure and AUC). F-measure displays the trade-off between the performances of the classifier. AUC measures the entire area under the entire Receiver Operating Characteristic (ROC) curve. The used metrics for within-project and cross-project are shown in Table 10. For a fair comparison, we selected the studies that apply F-measure together. Then the studies that apply AUC were chosen together. Some studies are excluded (highlighted in blue color); because St7 and St13 applied other evaluation metrics. The values in St21 were not presented in a clear way and St40 applied different metrics on the used tasks.

The highest values of F-measure for within-project and cross-project are shown in Figs. 9 and 10, respectively. The best F-measure for within-projects is 0.809, where the study St36 builds a deep learning model that uses Autoencoder. At the same time, the best F-measure for cross-projects is 0.736, where the study St22 builds a deep learning model that applies Attention Mechanism. The highest values of AUC for within-project and cross-project are shown in Fig. 11 and 12, respectively. The best AUC for within-projects is 0.892, where the study St19 builds a model that applies RNN. Additionally, the best AUC for cross-projects is 0.7, where the study St37 builds a deep learning model that uses DNN.

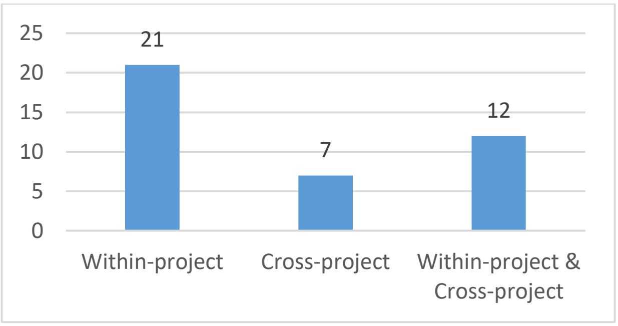

Fig. 3: Kinds of project level

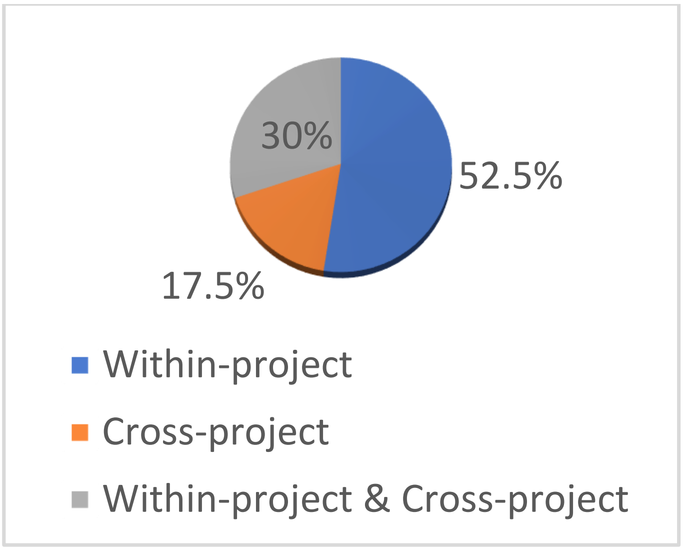

Fig. 4: The percentage of each project level

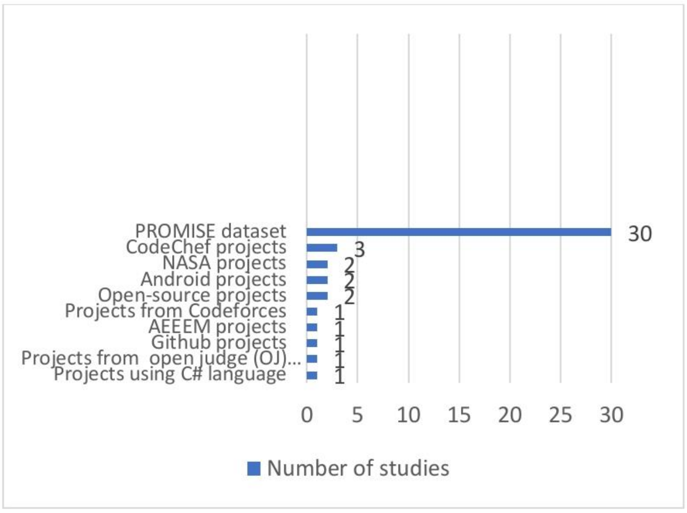

Fig. 5: Number of studies for each project in the datasets

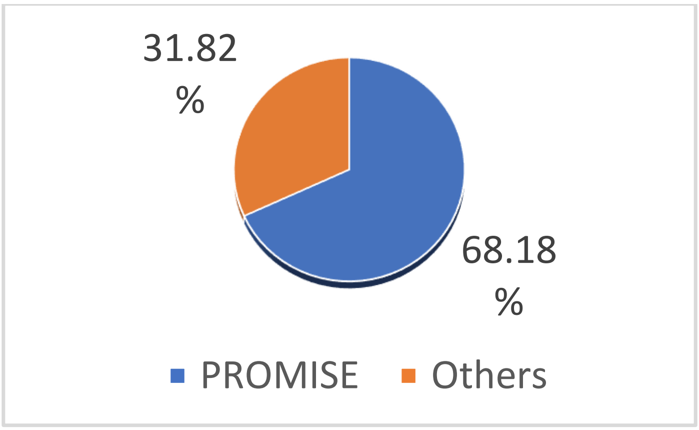

Fig. 6: The percentage of the used datasets

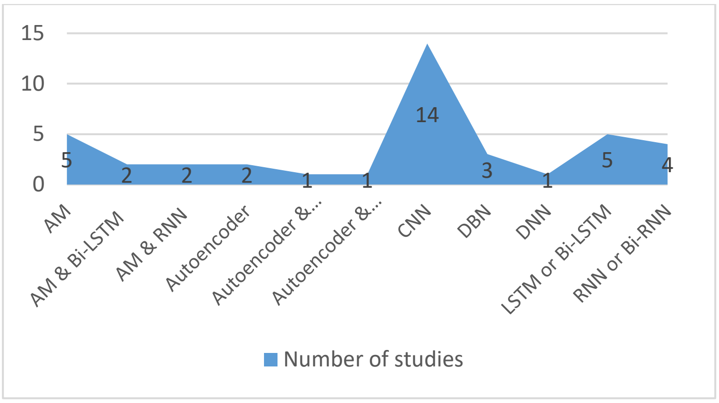

Fig. 7: Number of studies for each DL model

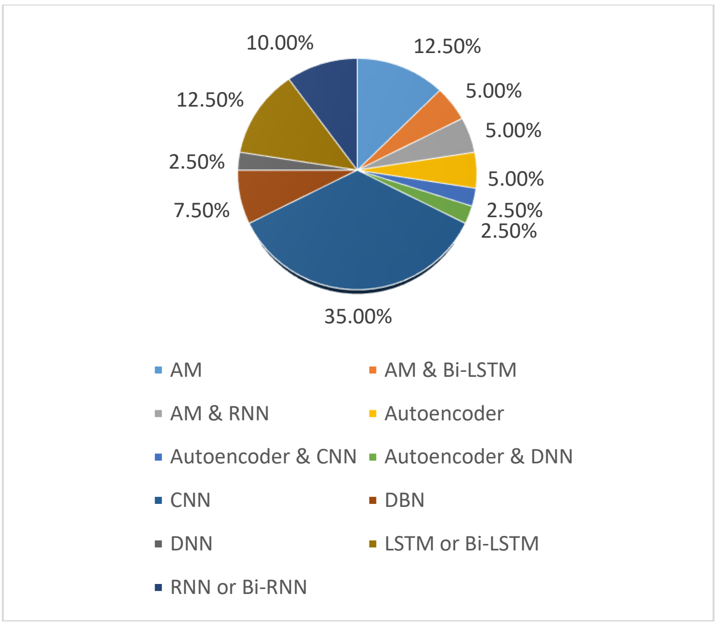

Fig. 8: Percentage of each DL model

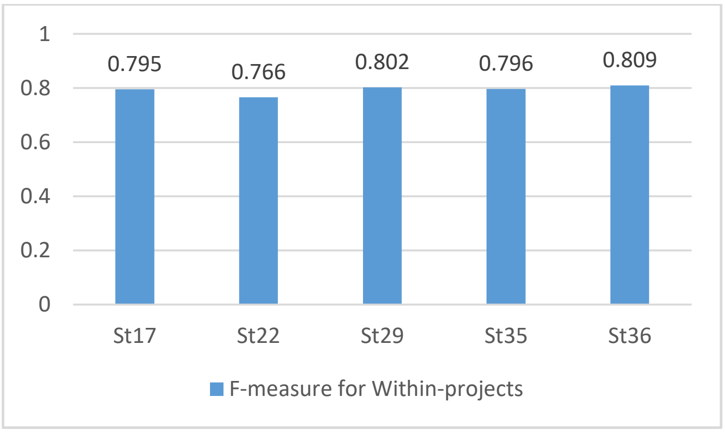

Fig. 9: F-measure for within-projects

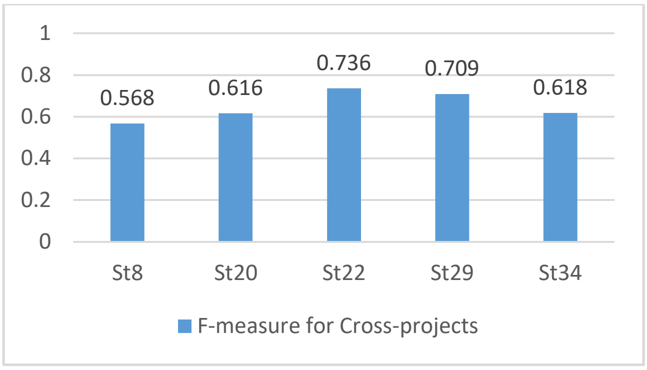

Fig. 10: F-measure for cross-projects

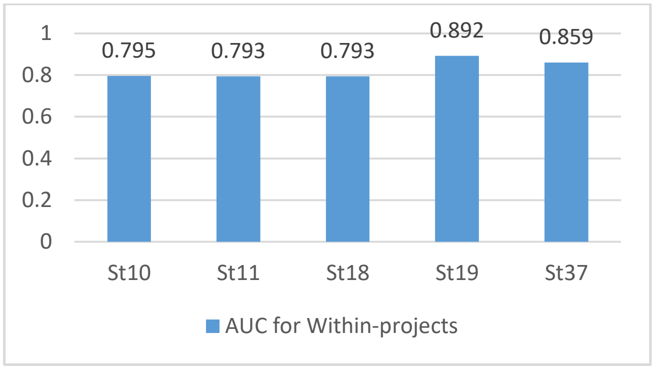

Fig. 11: AUC for within-projects

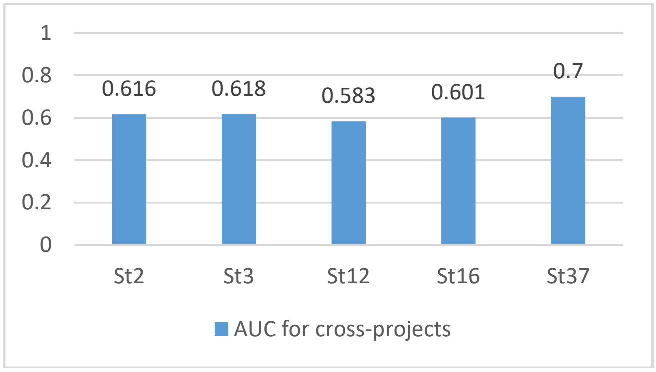

Fig. 12: AUC for cross-projects

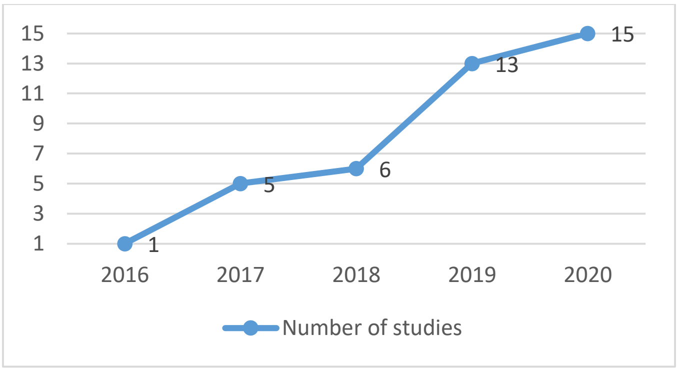

Fig. 13: The distribution of the studies over years

Table 8: The used datasets in the primary studies

Study	Dataset
St1	Camel, Forrest, Ivy, Jedit, Log4j, Lucene, Poi, Synapse, Velocity, Xalan, Xerces
St2	Ant, Camel, Ivy, Log4j, Lucene, Synapse, Velocity, Xalan, Xerces
St3	Camel, Forrest, Ivy, Log4j, Lucene, Poi, Synapse, Velocity, Xalan, Xerces
St4	Camel, Lucene, Poi, Synapse, Xalan, Xerces
St5	Ant, Camel, Ivy, Jedit, Log4j, Lucene, Poi, Synapse, Xalan, Xerces
St6	Ant, Camel, Ivy, Jedit, Log4j, Lucene, Pbeans, Poi, Synapse, Velocity, Xalan, Xerces
St7	Projects retrieved from a Pedagogical Programming Open Judge (OJ) system
St8	Ant, Camel, Jedit, Log4j, Lucene, Xalan, Xerces, Ivy, Synapse, Poi
St9	C# projects
St10	CodeChef projects
St11	CodeChef projects
St12	Ant, Camel, Jedit, Log4j, Lucene, Poi, Synapse, Velocity, Xalan, Xerces
St13	Ant, Camel, Ivy, Log4j, Lucene, Synapse, Velocity, Xalan, Xerces
St14	Ant, Camel, Ivy, Jedit, Log4j, Lucene, Poi, Synapse, Xalan, Xerces
St15	Camel, Jedit, Lucene, Poi, Synapse, Xalan, Xerces
St16	Ant, Camel, Jedit, Log4j, Lucene, Poi, Synapse, Velocity, Xalan, Xerces
St17	Codeforces projects
St18	CodeChef projects
St19	Ant, Camel, Ivy, Jedit, Log4j, Lucene, Poi, Synapse, Xalan, Xerces
St20	Camel, Ivy, Log4j, Lucene, Poi, Synapse, Xalan, Xerces
St21	Ant, Camel, Ivy, Jedit, Log4j, Lucene, Poi, Synapse, Xalan, Xerces
St22	Ant, Camel, Ivy, Jedit, Log4j, Lucene, Poi, Synapse, Xalan, Xerces
St23	Camel, Jedit, Log4j, Lucene, Poi, Synapse, Xalan, Xerces, GitHub projects
St24	Camel, Jedit, Lucene, Poi, Synapse, Xalan, Xerces
St25	Camel, Jedit, Lucene, Poi, Synapse, Xalan, Xerces
St26	Camel, Jedit, Lucene, Log4j, Poi, Xalan, Xerces
St27	Camel, Jedit, Lucene, Poi, Synapse, Xalan, Xerces
St28	Ant, Camel, Ivy, Jedit, Log4j, Lucene, Poi, Synapse, Xalan, Xerces
St29	Open-source projects
St30	Android projects
St31	Ant, Camel, Ivy, Jedit, Log4j, Lucene, Pbeans, Poi, Synapse, Velocity, Xalan, Xerces
St32	Camel, Forrest, Ivy, Log4j, Lucene, Poi, Synapse, Velocity, Xalan, Xerces
St33	Ant, Camel, Ivy, Jedit, Lucene, Poi, Synapse, Velocity, Xalan, Xerces
St34	Ant, Camel, Ivy, Jedit, Log4j, Lucene, Poi, Synapse, Xalan, Xerces
St35	Ant, Camel, Ivy, Jedit, Poi, Synapse, Xalan, Xerces, NASA projects
St36	Ant, Ivy, Jedit, Poi, Xerces, NASA projects
St37	Android projects
St38	Jedit, Log4j, Lucene, Poi, Synapse, Velocity, Xalan, Xerces
St39	Ant, Camel, Ivy, Jedit, Log4j, Lucene, Poi, Synapse, Xalan, Xerces, AEEEM projects
St40	Open-source projects

Table 9: DL models for each primary study

DL model	Study
AM	St5, St22, St27, St34, St40
AM and Bi-LSTM	St16, St17
AM and RNN	St15, St24
Autoencoder	St33, St36
Autoencoder and CNN	St29
Autoencoder and DNN	St35
CNN	St1, St2, St3, St4, St6, St7, St10,St11, St13, St18, St20, St25, St32, St39
DBN	St8, St14, St30
DNN	St37
LSTM or Bi-LSTM	St12, St21, St23, St26, St38
RNN or Bi-RNN	St9, St19, St28, St31

Table 10: F-measure and AUC for each primary study

	F-measure		AUC	
Study	Within-project	Cross-project	Within-project	Cross-project
St1	-	0.494	-	-
St2	-	-	-	0.616
St3	-	0.527	-	0.618
St4	0.560	-	-	-
St5	0.666	-	0.745	-
St6	0.618	-	-	-
St7	-	-	-	-
St8	0.641	0.568	-	-
St9	0.374	-	0.686	-
St10	0.741	-	0.795	-
St11	-	-	0.793	-
St12	-	-	-	0.583
St13	-	-	-	-
St14	0.641	0.539	-	-
St15	0.582	-	-	-
St16	-	-	-	0.601
St17	0.795	-	-	-
St18	0.731	-	0.793	-
St19	0.657	-	0.892	-
St20	0.669	0.616	-	-
St21	-	-	-	-
St22	0.766	0.736	-	-
St23	0.533	0.262	-	-
St24	0.564	-	0.738	-
St25	0.608	-	-	-
St26	0.521	-	-	-
St27	0.626	-	-	-
St28	0.590	0.548	-	-
St29	0.802	0.709	-	-
St30	-	-	0.780	-
St31	0.374	-	0.686	-
St32	-	0.532	-	-
St33	0.539	0.503	-	-
St34	0.642	0.618	-	-
St35	0.796	-	-	-
St36	0.809	-	-	-
St37	-	-	0.859	0.700
St38	-	-	0.637	-
St39	0.587	0.561	-	-
St40	-	-	-	-

E. Frameworks

Previous studies focused on traditional features such as code complexity, etc. However, these features failed to capture the semantics of source code. In this SLR, 40 primary studies are included. So, all the developed frameworks are analyzed based on the used deep learning model. Wang *et al.* (2016) introduced the DBN model to automatically extract semantic features of source code. Then fed them into the classifier to predict the defective code for both WPDP and CPDP. Edit Distance Similarity Computation algorithm was applied to identify the distances among token sequences. Also, Closest List Noise Identification (CLNI) was applied to remove the incorrectly labeled data. The framework was compared with PROMISE features and AST features for WPDP. For CPDP, it was compared with TCA+ (Nam *et al.*, 2013). TCA+ is an extended Transfer Component Analysis (TCA). The proposed framework improved the WPDP by 14.7, 11.5 and 14.2% in Precision, Recall and F-measure, respectively, compared to traditional features. For CPDP, it improved by 8.9% in F-measure compared to TCA+. Also, they worked on their study (Wang *et al.*, 2016) and proposed the enhanced DBN model (Wang *et al.*, 2018) to learn the semantic features from both source code and code changes. The same framework was used with the same evaluation metrics, but a new metric was added called PofB20 (Jiang *et al.*, 2013), to measure the percentage of defects by reviewing the top 20 of code lines. The proposed framework achieved better PofB20 than both TCA+ and the baseline.

However, existing studies on software defect prediction were limited to source code. So, Dong *et al.* (2018) proposed the DNN model to extract semantic features for Android projects that is called Smali2vec. DNN was used to build a supervised-learning model for the defect prediction. Based on that, Smali2vec improved WPDP by 85.98% in AUC and improved CPDP by 70%. Also, Fan *et al.* (2018) proposed the DBN model that applied on Android projects but it extracted both of functional and semantic features to improve the accuracy of the prediction.

In most studies, CNN achieved better performance than DBN and DNN, since it could capture context and semantics more effectively. To benefit from the hand-crafted features, (Li *et al.*, 2017) proposed the Defect Prediction via Convolutional Neural Network (DP-CNN). They combined the hand-crafted features with semantic features generated from AST to extract the best feature representation of source code. The combined features were used as input to Logistic Regression (LR) (Felix and Lee, 2020) classifier. Then the probability for each source code file was produced to predict if the file was defective or clean. To handle the imbalanced data, the defective files was duplicated many times to reach a balanced dataset. The framework was compared with the following baselines: DBN, traditional features method, DBN+ (an improved version of DBN proposed by them), the semantic features were combined with the traditional features) and CNN (CNN-learned features were fed to the classifier without combining traditional features). The proposed framework improved the selected baseline models by 12% on average of F-measure.

Pan *et al.* (2019) built their work depending on Li *et al.* (2017), they enhanced the capability of extracting global patterns and enhanced the model for better generalization. The framework was applied on the Simplified PROMISE Source Code (SPSC) dataset and PROMISE Source Code (PSC) dataset. SPSC dataset was smaller than the PSC dataset. For SPSC, the results showed that the proposed framework improved the existing CNN model by 2.2% on average of F-measure. For PSC, the results showed that the proposed framework improved the F-measure by 6%, the G-measure (Herbold *et al.*, 2017) by 5% and Matthews Correlation Coefficient (MCC) (Boughorbel *et al.*, 2017) by 2%.

Meilong *et al.* (2020) focused on learning more defect feature information from source code. The performance of the model increased when the filter length was 10 andthe optimal number of filters was 20. Furthermore, Huo *et al.* (2018) embedded code comments automatically to generate semantics from source code. Code comments were capable of generating more semantic features. The framework was applied on both WPDP and CPDP. For WPDP, it was compared with the following baselines: LR, Naive Bayes (NB) (Hammouri *et al.*, 2018), ADTree (Tan *et al.*, 2015) and DBN. For CPDP, DBN and TCA+ were used in the comparison. The results ensured that code comments were capable of generating more semantic features.

Phan *et al.* (2017) applied the multi-view multi-layer directed graph-based convolutional neural network to learn semantic features from Control Flow Graphs (CFGs) (Kanuparthi*et al.*, 2016) of programs. The framework was compared with feature-based and tree-based approaches. It improved the accuracy from 4.08 to 15.49% compared to the feature-based approach and from 1.2 to 12.39% compared with the tree-based approaches. In addition, they introduced the Tree-Based Convolutional Neural Networks (TBCNN) framework (Phan *et al.*, 2018a), it enhanced the time and accuracy of classifiers by applying several pruning tree models. TBCNN achieved a better accuracy by 92.63% on average. Also, they presented the Convolutional Neural Network on Assembly Code (ASCNN) framework (Phan and Le Nguyen, 2017), it applied the multi-view convolutional neural network to learn defect features from the assembly code instead of AST. ASCNN achieved results better than the selected baselines based on software metrics and Abstract Syntax Trees. They modified their work (Phan *et al.*, 2017) and introduced a new model (Phan *et al.*, 2018b). The task of malware analysis was added to check if the executable file was malware or not.

To learn the transferable joint features, (Qiu *et al.*, 2019a) used the TCA algorithm. TCA handled data and the labeled data were fed to LR classifier to predict if the new project was defective. The proposed framework improved TCA by 30.1% and CNN by 62.8% on average of MCC. Also, Cai *et al.* (2019) employed the hierarchical SoftMax to reduce the time complexity of adjusting the distributed representation of context words. Sheng *et al.* (2020) extracted the transferable semantic features from source code. It applied the adversarial discriminative learning on source project and target project.

To handle the data distribution variance between source and target projects, Deng *et al.* (2020a) proposed the Multi-Kernel Transfer Convolutional Neural Network (MK-TCNN) for CPDP. MK-TCNN achieved a better F-measure by 0.494 on average, compared with the other baselines. Qiu *et al.* (2019b) added the matching layer to mine the transferable semantics by decreasing classification error and distribution variance between source and target project. The proposed framework achieved a better F-measure by 0.532 on average, compared with the selected baselines. Besides, Shi *et al.* (2021) represented the code by different representations. The most crucial information of nodes was coded. The serialization was applied to order the nodes and the AST was coded into an embedding sequence. For WPDP, the framework improved DP-CNN by 2.03% on average of F-measure. For CPDP, improved DP-CNN by 5.44% on average of F-measure.

Wen *et al.* (2018) proposed the RNN model that is called FENCES. It used fine-grained change analysis to extract the change sequences. FENCES improved the average of F-measure from 31.6 to 46.8% and the average of AUC from 4.2 to 16.1%, compared to the prediction models built on the traditional metrics. Tian and Tian (2020) applied the Gated

Recurrent Unit (GRU) (Dey and Salem, 2017) on the vectors to extract semantic features. The proposed framework improved WPDP by 11.0% on average of F-measure. For CPDP, it improved by 10.4% on average of F-measure compared to tree-LSTM model. Zhang *et al*. (2018b) used the RNN model to learn the regularities in the source code. Then the Cross-Entropy features were extracted and combined with the traditional features. Cross-Entropy helped in extracting the implicit knowledge in software repositories. The proposed framework's performance improved by 2.8% on average of F-measure. Nivetha and Kavitha (2019) combined the Cross-Entropy metric and Abstract Syntax Tree to be used as a new code metric. It produced better accurate defect prediction and the new metric achieved better results than the existing metrics. The results showed that the proposed framework improved Precision by 4.8%, Recall by 2.4% and F-measure by 8.5% on average.

Dam *et al*. (2019) proposed the deep tree-based LSTM model, the model took a raw AST of a source file to be used as input. The model used two classifiers (LR and Random Forest (RF) Zhang *et al*., 2018a), to choose the best one of them. For WPDP, the model that used RF achieved good results with all four metrics (F-measure, Precision, Recall and AUC). For CPDP, the model achieved very high Recall across all projects. Liang *et al*. (2019) combined word embedding with the LSTM model for defect prediction. The mapping table was used to map each token to real-valued vector. The vectors and their labels were used to build LSTM. For WPDP, the framework improved DBN, tree-based LSTM (tb-LSTM) and Improved Subclass Discriminant Analysis (ISDA) by 8.2, 4.3 and 8.4% on average of F-measure, respectively. For CPDP, it improved them by 8.5, 0.8 and 1.5% on average of F-measure, respectively. Deng *et al*. (2020b) applied their LSTM model to automatically learn semantic features from source code. Also, Zhou and Lu (2020) introduced the Bi-LSTM model to extract semantic features from source code.

To automatically construct meaningful vector representations for token sequences, Li *et al*. (2019) proposed the CPDP approach that focused on the Simplified AST (S-AST). The project-independent node type only was remained and other project-specific information was ignored. CPDP approach was compared with the following baselines (Li17-CNN, CamargoCruz09-DT, Turhan09-DT, Menzies11-RF and Watanabe08-DT). The approach improved the baselines by 3.00, 17.54, 8.77, 14.76 and 8.97%, respectively, on average AUC.

Shi *et al*. (2020) built their work based on code2vec, they proposed the PathPair2Vec framework based on Attention Mechanism. The different parts of the terminal node were encoded. Then path pairs were applied to identify the semantics of source code. PathPair2Vec achieved F-measure better than DP-CNN. It exceeded DP-CNN by 17.88%. Hoang *et al*. (2020) proposed the neural network model to learn the representation of code changes. It used the Attention Mechanism and multiple comparison functions to identify the difference between the added and removed code. It was applied to the following tasks: Log message generation, bug fixing patch identification and just-in-time defect prediction. It achieved better performance for all the tasks. Zhang and Wu (2020) proposed the software defect prediction via transformer. It automatically extracted semantic features by the encoder. Then, it used scaled dot-product attention and multi-head attention mechanism to capture the key features. It improved CNN by 8% on average of F-measure and improved RNN by 7%.

Humphreys and Dam (2019) solved the long dependency problem by allowing each token to access other tokens through only one connection with Self-Attention. The model was compared with the model of Wang *et al*. (2016). The proposed model achieved 0.666 on average of F-measure, while Song Wang *et al*.'s model achieved 0.641 only. Chen *et al*. (2020) combined the Self-Attention Mechanism with the Deep Transfer Learning model. It visualized program files of source code as images. Self-Attention Mechanism was applied to capture the semantic features of the images. Transfer Learning model was used to reduce the difference in sample distributions between projects. The framework improved TCA+, DBN, LSTM and CNN by 29.0, 8.8, 24.8 and 15.5% on average of F-measure, respectively.

To capture the key features of defects, Fan *et al*. (2019a) proposed the Defect Prediction via Attention-based Recurrent Neural Network (DP-ARNN) (Fan *et al*., 2019b) and Defect Prediction via Attention Mechanism (DP-AM) (Fan *et al*., 2019a) frameworks. The RNN was used to automatically extract semantic features from source code. Then, Attention Mechanism was used to capture the key features. DP-ARNN improved the selected baselines by 14% on average of F-measure and improved the average of AUC by 7%. DP-AM improved the selected baselines by 11% on average of F-measure. Chen *et al*. (2019) built their work depending on Li *et al*., (2019), Attention Mechanism was added over the Bi-LSTM layer to learn the weight of the vectors from the learned semantic features.

To handle the problem of Time Limit Exceed (TLE), Zhou *et al*. (2019) built the DeepTLE model to predict if there were performance defects before running the test cases. DeepTLE saved 96% of the time cost and 82% of accuracy.

Wang and Lu (2020) applied the Convolutional Autoencoder (CAE) to learn semantic features by decreasing the reconstruction error between input and output. Domain Adaptation was applied for CPDP to improve the transferability of CAE-based features.

Zhang *et al*. (2020) combined the Stacked Contractive Autoencoder (SCAE) and the Multi-Objective Defect Prediction Model (SMONGE). SCAE was used to learn the more robust features. SMONGE was used to utilize the multi-objective NSGAII algorithm to optimize the Extreme Learning Machine (ELM). SCAE achieved 13.25, 21.99 and 43.39% on average of F-measure, G-measure and MCC compared with the used feature extraction methods. SMONGE achieved 9.09, 4.84 and

14.79% on average of F-measure, G-measure and MCC compared with the used defect predictors.

Zhu *et al*. (2020a) proposed the just-in-time defect prediction model based on Denoising Autoencoder (DAE) and CNN. CNN was used to extract semantic features, then DAE was used to learn the more robust features. Zhu *et al*. (2020b) used the DAE to learn the more robust features and DNN to learn the semantic features. The model achieved 8.96, 34.83 and 11.35% on average of F-measure, MCC and G-measure compared with the used feature extraction methods. In terms of PF, it achieved a lower value than other methods. The framework achieved 11.31, 46.55 and 15.08% on average of F-measure, MCC and G-measure compared to the used defect predictors.

F. Number of Publications

In this SLR, 40 primary studies published in software defect prediction that use deep learning models. After analyzing the studies, we find that all studies have published since 2016. Also, the number of studies is increased over the years. Figure 13 shows the distribution of the studies over the years.

Discussion

In this study, we have reviewed Forty primary studies on software defect prediction using deep learning models that were published by 15 December 2020. We provide a summary of software defect prediction models and identify the used datasets, framework and evaluation metrics of the proposed models. The data are collected from all available research studies in the selected digital libraries. A software defect is a fault in a document, code, software, or system that leads to producing an incorrect behavior during execution, causing failure to be appeared. The system that contains defects lead to many problems, including loss of money and time of fixing defects.

Following the predefined research questions in Section II, the first question is related to the kind of project levels that researchers often use to build a software defect prediction model. Within-project has applied in most of the primary studies. However, some models achieve good performance in within-project and bad results with cross-project. So, the model should be applied on both within-project and cross-project to make sure of the performance of the model. Unfortunately, only 30% of the primary studies have applied both of within-project and cross-project.

The second question is related to the datasets that are often used for software defect prediction. Most of the primary studies have applied PROMISE repository only. However, applying many datasets and comparing the results among them is a good indicator for the effectiveness of the model. And also, public datasets have some issues such as lack of data quality. Therefore, the results of the proposed models may be inaccurate.

The third question is related to the models of deep learning that are often used for software defect prediction. We find that some studies use one of the deep learning models and other studies use a combination of deep learning models. And also, the most used deep learning model in the primary studies is CNN. Moreover, the hybrid models have achieved good results compared to individual deep learning models.

The fourth question is related to the evaluation metrics. These metrics are used to compare the proposed model with the other selected baseline models. Most of the primary studies have applied only one evaluation metric. However, using set of evaluation metrics is better than using F-measure or AUC only. We need to apply many evaluation metrics to validate the applicability of the model. In addition to check the performance of the proposed model.

The fifth question is related to the frameworks that are used in software defect prediction. All the frameworks are presented in details in section III. Each primary study describes the steps of the proposed model. It presents how to collect the data from the selected datasets and also, it identifies how to extract the semantics of the source code. Then, the proposed model is compared with the selected baseline models to evaluate the performance of the framework.

The last question is related to the number of publications in software defect prediction using deep learning models. This SLR is not limited to specific time period. However, all the primary studies have published since 2016. And also, the number of the studies is increased over the years.

In this SLR, we raise several challenges related to software defect prediction and propose the practices that can be performed to overcome these challenges. The first challenge is using public datasets such as PROMISE and NASA in software defect prediction models. There are some issues in public datasets, which can lead to poor prediction of software defects. We recommend to apply the techniques of data preprocessing to enhance the quality of public datasets.

Another challenge is related to building software defect prediction models. There are various deep learning models that are used to make the prediction of software defects. A few studies have applied hybrid models, which achieve better performance and high prediction rates compared to individual deep learning models. We need to apply more hybrid models, such as using CNN with Bi-LSTM. We recommend to increase the number of hybrid models for improving software defect prediction.

And also, another challenge is related to classifying the files of the source code as defective or clean. However, it is better to know more information about the number of defects and also the severity and priority of each defect.

Study Limitations

This SLR has some limitations. First, we have used the predefined search string to find the relevant primary studies on software defect prediction. And also, the studies are related to deep learning models only. Second, the primary studies are retrieved based on the selected digital libraries. Moreover, English papers are selected only.

Conclusion and Future Work

This SLR aims to identify and analyze the datasets, models, frameworks and evaluation metrics used in software defect prediction using deep learning models. It presents the primary studies that concentrate on extracting semantic features of the source code. Forty primary studies are selected to be evaluated in this SLR. This SLR answers the six research questions that are identified in the literature review protocol. We summarize the main findings as follows:

- For project level, within-project was the most used in the primary studies by 52.5%
- The most used dataset was PROMISE by 68.18%
- Most of the primary studies applied individual deep learning models such as Convolutional Neural Network (CNN), Recurrent Neural Network (RNN), etc. Few research studies applied hybrid deep learning models such as Attention Mechanism with RNN
- The most applied model was CNN by 35%
- F-measure and Area Under the Curve (AUC) were the most used evaluation metrics
- The summary of the primary studies is shown in Table 11. Also, the map of the primary studies that is designed by VOSviewer tool. The map is shown in Fig. 14. "VOSviewer offers text mining functionality that can be used to construct and visualize co-occurrence networks of important terms extracted from a body of scientific literature (VOSviewer, 2020)"

Table 11: Summary of the primary studies

Study	Authors	Journal / Conference	Project level	Datasets	Model	Evaluation metrics	Framework	Year
St1	Deng *et al.*	IEEE ACCESS	Cross	PROMISE	CNN	F-measure	MK-TCNN	2020a
St2	Cai *et al.*	IEEE ACCESS	Cross	PROMISE	CNN	AUC	TBCNN-THFL	2019
St3	Sheng *et al.*	IEEE ACCESS	Cross	PROMISE	CNN	F-measure, AUC, PofB20	ADCNN	2020
St4	Meilong *et al.*	Hindawi Mathematical Problems in Engineering	Within	PROMISE	CNN	F-measure	SDP-S2S	2020
St5	Humphreys and Dam	International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering	Within	PROMISE	AM	Precision, Recall, F-measure, AUC	Self-attention Transformer Encoder	2019
St6	Pan *et al.*	MDPI applied sciences	Within	PROMISE	CNN	F-measure, G-measure, MCC	Improved CNN	2019
St7	Phan *et al.*	Data and Knowledge Engineering	Within	pedagogical open judge (OJ) system	CNN	Accuracy	TBCNN	2017
St8	Wang *et al.*	IEEE International Conference on Software Engineering	Within and Cross	PROMISE	DBN	Precision, Recall, F-measure	DBN Model	2016
St9	Nivetha and Kavitha	International Journal of Engineering and Advanced Technology	Within	C# projects	Bi-RNN	Precision, Recall, F-measure, AUC	RNNLM	2019
St10	Phan *et al.*	Asia Pacific Symposium on Intelligent and Evolutionary Systems	Within	CodeChef projects	CNN	Accuracy, F-measure, AUC	ASCNN	2017
St11	Phan *et al.*	International Conference on Tools with Artificial Intelligence	Within	CodeChef projects	CNN	Accuracy, AUC	DGCNN	2017
St12	Li *et al*.	International Joint Conference on Neural Networks	Cross	PROMISE	Bi-LSTM	AUC	CPDP Approach	2019
St13	Qiu *et al.*	DOI reference number: 10.18293/SEKE2019-070	Cross	PROMISE	CNN	MCC	CNN-THFL	2019a
St14	Wang *et al.*	IEEE Transactions on Software Engineering	Within & Cross	PROMISE	DBN	Precision, Recall, F-measure	DBN-based Feature Generation	2018
St15	Fan *et al.*	Asia-Pacific Software Engineering Conference	Within	PROMISE	AM and RNN	F-measure	DP-AM	2019a
St16	Chen *et al*.	IEEE ACCESS	Cross	PROMISE	AM & Bi- LSTM	AUC	DeepCPDP	2019
St17	Zhou *et al*.	Asia-Pacific Software Engineering Conference	Within	Codeforces projects	AM & Bi-LSTM	Accuracy, Recall, F-measure	DeepTLE	2019
St18	Phan *et al.*	Neural Networks	Within	CodeChef projects	CNN	Accuracy, F-measure, AUC	DGCNN	2018a
St19	Wen *et al.*	IEEE Transactions on Software Engineering	Within	PROMISE	RNN	Precision, Recall, F-measure, AUC	FENCES	2018
St20	Huo *et al.*	IEEE International Conference on Data Mining	Within and Cross	PROMISE	CNN	F-measure	CAP-CNN	2018
St21	Dam *et al*.	2019 IEEE/ACM International Conference on Mining Software Repositories	Within & Cross	PROMISE	LSTM	Precision, Recall, F-measure, AUC	Deep Tree-based LSTM	2019
St22	Shi *et al.*	Journal of Computer Languages	Within & Cross	PROMISE	AM	F-measure	PathPair2Vec	2020
St23	Hongliang *et al*.	IEEE ACCESS	Within & Cross	PROMISE and GitHub projects	LSTM	Precision, Recall, F-measure	Seml	2019
St24	Fan *et al.*	Hindawi Scientific Programming	Within	PROMISE	AM and RNN	F-measure, AUC	DP-ARNN	2019b
St25	Li *et al*.	International Conference on Software Quality, Reliability and Security	Within	PROMISE	CNN	F measure	DP-CNN	2017
St26	Deng *et al.*	IET Software	Within	PROMISE	LSTM	F measure	DP-LSTM	2020b
St27	Zhang and Wu	Information Technology Networking Electronic and Automation Control Conference	Within	PROMISE	AM	F-measure	DP-Transformer	2020
St28	Tian and Tian	National Natural Science Foundation of China, IEEE	Within and Cross	PROMISE	RNN	Precision, Recall, F-measure	Model based on program slice	2020
St29	Zhu *et al*.	IET Software	Within and Cross	Open-source projects	AE and CNN	Accuracy, Precision, Recall, F-measure, PofB20	DAECNN-JDP	2020a
St30	Fan *et al*.	IEEE International Conference on Computer Software & Applications	Within	Android projects	DBN	AUC	HIRER	2018
St31	Zhang *et al.*	IEEE International Conference on Software Quality, Reliability and Security	Within	PROMISE	RNN	Precision, Recall, F-measure, AUC	DefectLearner	2018a
St32	Qiu *et al.*	Applied Sciences	Cross	PROMISE	CNN	F-measure	TCNN	2019b
St33	Wang and Lu	DOI reference number: 10.18293/SEKE2020-036	Within & Cross	PROMISE	AE	F-measure	SCAE	2020
St34	Chen *et al.*	ICSE 2020, ACM	Within & Cross	PROMISE	AM	F-measure	DTL-DP	2020
St35	Zhu *et al*.	Computers, Materials & Continua	Within	PROMISE and NASA projects	AE and DNN	F-measure, MCC, PF, G-measure	SL-Isomap and DLDD	2020b
St36	Zhang *et al.*	Computers, Materials & Continua	Within	PROMISE and NASA projects	AE	F-measure, G-measure, PF, MCC	SCAE & SMONGE	2020
St37	Dong *et al.*	Wireless Pers Commun, Springer	Within & Cross	Android projects	DNN	AUC	Smali2vec	2017
St38	Zhou and Lu	International Conference on Software Quality, Reliability and Security	Within	PROMISE	Bi-LSTM	AUC, MCC	LSTM-BT	2020
St39	Shi *et al.*	Software: Evolution and Process	Within and Cross	PROMISE and AEEEM projects	CNN	F-measure	MPT-embedding	2020
St40	Hoang *et al*.	arXiv, ACM	Within	Open-source projects	AM	AUC	CC2Vec	2020

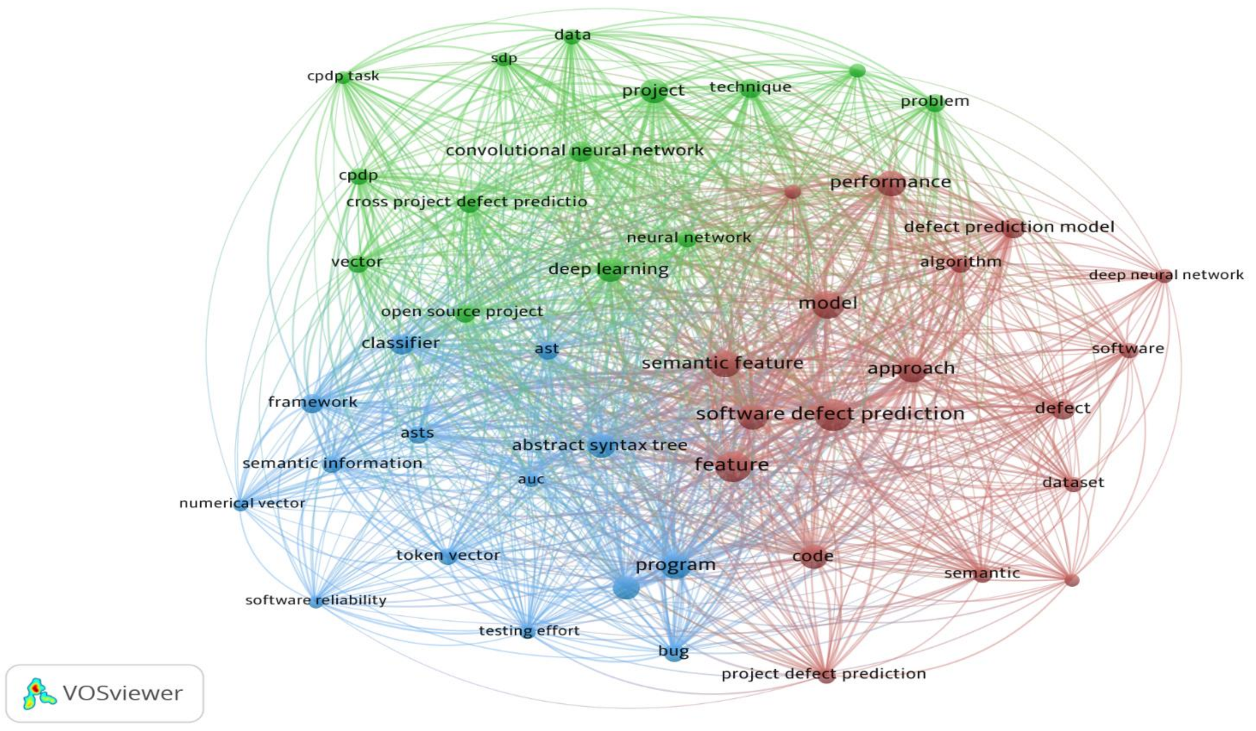

Fig. 14: The map of the primary studies

For future studies on software defect prediction using deep learning models, the following are recommended:

- The hybrid deep learning models achieve good results compared to the individual models. There are some hybrid models can be applied in the future such as CNN with Bi-LSTM. CNN can capture the semantics of the source code more effectively and Bi-LSTM can detect the information of long-term dependencies
- More data preprocessing techniques should be applied to improve the quality of public datasets
- Recently, most of the organizations apply the agile methodology. Therefore, deep learning models should be applied for the agile methodology

Author's Contributions

Ahmed Bahaa: Reviewing, supervising, revising manuscript contents and editing manuscript.

Enas Mohamed Fathy: Collecting, synthesizingrelevant literature, and drafting manuscript contents.

Ahmed Sharaf Eldin: Supervising manuscript contents.

Laila A. Abd-Elmegid: Reviewing, supervising and revising manuscript contents.

Ethics

We confirm that we have read and approved the manuscript and no ethical issues involved.

References

Olsen, K. (2019). International Software Testing Qualifications Board. Certified Tester, Foundation Level, Version.

Boughorbel, S., Jarray, F., & El-Anbari, M. (2017). Optimal classifier for imbalanced data using Matthews Correlation Coefficient metric. PloS One, 12, e0177678. https://doi.org/10.1371/journal.pone.0177678

Cai, Z., Lu, L., & Qiu, S. (2019). An abstract syntax tree encoding method for cross-project defect prediction. IEEE Access, 7, 170844-170853. https://doi.org/10.1109/ACCESS.2019.2953696

Chen, D., Chen, X., Li, H., Xie, J., & Mu, Y. (2019). Deepcpdp: Deep learning based cross-project defect prediction. IEEE Access, 7, 184832-184848. https://doi.org/10.1109/ACCESS.2019.2961129

Chen, J., Hu, K., Yu, Y., Chen, Z., Xuan, Q., Liu, Y., & Filkov, V. (2020, June). Software visualization and deep transfer learning for effective software defect prediction. In Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering (pp. 578-589). https://doi.org/10.1145/3377811.3380389

Dahou, A., Elaziz, M. A., Zhou, J., & Xiong, S. (2019). Arabic sentiment classification using convolutional neural network and differential evolution algorithm. Computational Intelligence and Neuroscience, 2019. https://doi.org/10.1155/2019/2537689

Dam, H. K., Pham, T., Ng, S. W., Tran, T., Grundy, J., Ghose, A., ... & Kim, C. J. (2019, May). Lessons learned from using a deep tree-based model for software defect prediction in practice. In 2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR) (pp. 46-57). IEEE. https://ieeexplore.ieee.org/abstract/document/8816787/

Deng, J., Lu, L., & Qiu, S. (2020a). Software defect prediction via LSTM. IET Software, 14(4), 443-450. https://doi.org/10.1049/iet-sen.2019.0149

Deng, J., Lu, L., Qiu, S., & Ou, Y. (2020b). A Suitable AST Node Granularity and Multi-Kernel Transfer Convolutional Neural Network for Cross-Project Defect Prediction. IEEE Access, 8, 66647-66661. https://doi.org/10.1109/ACCESS.2020.2985780

Dey, R., & Salem, F. M. (2017, August). Gate-variants of gated recurrent unit (GRU) neural networks. In 2017 IEEE 60th international midwest symposium on circuits and systems (MWSCAS) (pp. 1597-1600). IEEE. https://doi.org/10.1109/MWSCAS.2017.8053243

Dong, F., Wang, J., Li, Q., Xu, G., & Zhang, S. (2018). Defect prediction in android binary executables using deep neural network. Wireless Personal Communications, 102(3), 2261-2285. https://doi.org/10.1007/s11277-017-5069-3

Fan, G., Diao, X., Yu, H., Yang, K., & Chen, L. (2019a). Software defect prediction via attention-based recurrent neural network. Scientific Programming, 2019. https://doi.org/10.1155/2019/6230953

Fan, G., Diao, X., Yu, H., Yang, K., & Chen, L. (2019b, December). Deep Semantic Feature Learning with Embedded Static Metrics for Software Defect Prediction. In 2019 26th Asia-Pacific Software Engineering Conference (APSEC) (pp. 244-251). IEEE. https://ieeexplore.ieee.org/abstract/document/8946058/

Fan, Y., Cao, X., Xu, J., Xu, S., & Yang, H. (2018, July). High-Frequency Keywords to Predict Defects for Android Applications. In 2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC) (Vol. 2, pp. 442-447). IEEE. https://ieeexplore.ieee.org/abstract/document/8377901/

Felix, E. A., & Lee, S. P. (2020). Predicting the number of defects in a new software version. PloS One, 15(3), e0229131. https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0229131

Ferenc, R., Tóth, Z., Ladányi, G., Siket, I., & Gyimóthy, T. (2018, October). A public unified bug dataset for Java. In Proceedings of the 14th International Conference on Predictive Models and Data Analytics in Software Engineering (pp. 12-21). https://doi.org/10.1145/3273934.3273936

Hammouri, A., Hammad, M., Alnabhan, M., & Alsarayrah, F. (2018). Software bug prediction using machine learning approach. International Journal of Advanced Computer Science and Applications, 9(2), 78-83. https://doi.org/10.14569/IJACSA.2018.090212

Herbold, S., Trautsch, A., & Grabowski, J. (2017). A comparative study to benchmark cross-project defect prediction approaches. IEEE Transactions on Software Engineering, 44(9), 811-833. https://doi.org/10.1145/3180155.3182542

Hoang, T., Kang, H. J., Lo, D., & Lawall, J. (2020, June). CC2Vec: Distributed representations of code changes. In Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering (pp. 518-529). https://doi.org/10.1145/3377811.3380361

Hosseini, S., Turhan, B., & Gunarathna, D. (2017). A systematic literature review and meta-analysis on cross project defect prediction. IEEE Transactions on Software Engineering, 45(2), 111-147. https://ieeexplore.ieee.org/abstract/document/8097045

Huang, X., Zhang, H., Zhou, X., Babar, M. A., & Yang, S. (2018, May). Synthesizing qualitative research in software engineering: A critical review. In Proceedings of the 40th International Conference on Software Engineering (pp. 1207-1218). https://doi.org/10.1145/3180155.3180235

Humphreys, J., & Dam, H. K. (2019, May). An explainable deep model for defect prediction. In 2019 IEEE/ACM 7th International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering (RAISE) (pp. 49-55). IEEE. https://doi.org/10.1109/RAISE.2019.00016

Huo, X., Yang, Y., Li, M., & Zhan, D. C. (2018, November). Learning semantic features for software defect prediction by code comments embedding. In 2018 IEEE International Conference on Data Mining (ICDM) (pp. 1049-1054). IEEE. https://doi.org/10.1109/ICDM.2018.00133

Jiang, T., Tan, L., & Kim, S. (2013, November). Personalized defect prediction. In 2013 28th IEEE/ACM International Conference on Automated Software Engineering (ASE) (pp. 279-289). IEEE. https://doi.org/10.1109/ASE.2013.6693087

Kanuparthi, A., Rajendran, J., & Karri, R. (2016, May). Controlling your control flow graph. In 2016 IEEE International Symposium on Hardware Oriented Security and Trust (HOST) (pp. 43-48). IEEE. https://doi.org/10.1109/HST.2016.7495554

Li, H., Li, X., Chen, X., Xie, X., Mu, Y., & Feng, Z. (2019, July). Cross-project Defect Prediction via ASTToken2Vec and BLSTM-based Neural Network. In 2019 International Joint Conference on Neural Networks (IJCNN) (pp. 1-8). IEEE. https://ieeexplore.ieee.org/abstract/document/8852135/

Li, J., He, P., Zhu, J., & Lyu, M. R. (2017, July). Software defect prediction via convolutional neural network. In 2017 IEEE International Conference on Software Quality, Reliability and Security (QRS) (pp. 318-328). IEEE.

Li, Z., & Liu, G. (2018, December). Optimizing attention mechanism for neural machine transltion. In 2018 IEEE 4th International Conference on Computer and Communications (ICCC) (pp. 2398-2404). IEEE. https://ieeexplore.ieee.org/abstract/document/8780734/

Li, Z., Jing, X. Y., & Zhu, X. (2018). Progress on approaches to software defect prediction. IET Software, 12(3), 161-175. https://doi.org/10.1049/iet-sen.2017.0148

Liang, H., Yu, Y., Jiang, L., & Xie, Z. (2019). Seml: A semantic lstm model for software defect prediction. IEEE Access, 7, 83812-83824. https://doi.org/10.1109/ACCESS.2019.2925313

Lin, G., Zhang, J., Luo, W., Pan, L., Xiang, Y., De Vel, O., & Montague, P. (2018). Cross-project transfer representation learning for vulnerable function discovery. IEEE Transactions on Industrial Informatics, 14(7), 3289-3297. https://ieeexplore.ieee.org/abstract/document/8329207

Meilong, S., He, P., Xiao, H., Li, H., & Zeng, C. (2020). An Approach to Semantic and Structural Features Learning for Software Defect Prediction. Mathematical Problems in Engineering, 2020. https://doi.org/10.1155/2020/6038619

Nam, J., Pan, S. J., & Kim, S. (2013, May). Transfer defect learning. In 2013 35th international conference on software engineering (ICSE) (pp. 382-391). IEEE. https://ieeexplore.ieee.org/abstract/document/6606584/

Nivetha. R., & Kavitha, S. (2019). Bidirectional Recurrent Neural Network Language Model: Cross Entropy Churn Metrics for Defect Prediction Modelling. International Journal of Engineering and Advanced Technology, pp, 2792-2800. https://doi.org/10.35940/ijeat.F8859.088619

Okoli, C. (2015). A guide to conducting a standalone systematic literature review. Communications of the Association for Information Systems, 37(1), 43. https://doi.org/10.17705/1CAIS.03743

Pan, C., Lu, M., Xu, B., & Gao, H. (2019). An improved CNN model for within-project software defect prediction. Applied Sciences, 9(10), 2138. https://doi.org/10.3390/app9102138

Phan, A. V., & Le Nguyen, M. (2017, November). Convolutional neural networks on assembly code for predicting software defects. In 2017 21st Asia Pacific Symposium on Intelligent and Evolutionary Systems (IES) (pp. 37-42). IEEE. https://doi.org/10.1109/IESYS.2017.8233558

Phan, A. V., Chau, P. N., Le Nguyen, M., & Bui, L. T. (2018a). Automatically classifying source code using tree-based approaches. Data & Knowledge Engineering, 114, 12-25. https://doi.org/10.1016/j.datak.2017.07.003 IEEE. https://doi.org/10.1109/ICTAI.2017.00019

Phan, A. V., Le Nguyen, M., Nguyen, Y. L. H., & Bui, L. T. (2018b). Dgcnn: A convolutional neural network over large-scale labeled graphs. Neural Networks, 108, 533-543. https://doi.org/10.1016/j.neunet.2018.09.001

Phan, A. V., Le Nguyen, M., & Bui, L. T. (2017, November). Convolutional neural networks over control flow graphs for software defect prediction. In 2017 IEEE 29th International Conference on Tools with Artificial Intelligence (ICTAI) (pp. 45-52).

Qiu, S., Lu, L., Cai, Z., & Jiang, S. (2019). Cross-Project Defect Prediction via Transferable Deep Learning-Generated and Handcrafted Features. In SEKE (pp. 431-552).

Qiu, S., Xu, H., Deng, J., Jiang, S., & Lu, L. (2019). Transfer convolutional neural network for cross-project defect prediction. Applied Sciences, 9(13), 2660. https://www.mdpi.com/2076-3417/9/13/2660

Sabir, F., Palma, F., Rasool, G., Guéhéneuc, Y. G., & Moha, N. (2019). A systematic literature review on the detection of smells and their evolution in object-oriented and service-oriented systems. Software: Practice and Experience, 49(1), 3-39. https://doi.org/10.1002/spe.2639

Samir, M., El-Ramly, M., & Kamel, A. (2019, November). Investigating the Use of Deep Neural Networks for Software Defect Prediction. In 2019 IEEE/ACS 16th International Conference on Computer Systems and Applications (AICCSA) (pp. 1-6). IEEE. https://doi.org/10.1109/AICCSA47632.2019.9035240

Sheng, L., Lu, L., & Lin, J. (2020). An adversarial discriminative convolutional neural network for cross-project defect prediction. IEEE Access, 8, 55241-55253. https://doi.org/10.1109/ACCESS.2020.2981869

Shi, K., Lu, Y., Chang, J., & Wei, Z. (2020). PathPair2Vec: An AST path pair-based code representation method for defect prediction. Journal of Computer Languages, 59, 100979. https://doi.org/10.1016/j.cola.2020.100979

Shi, K., Lu, Y., Liu, G., Wei, Z., & Chang, J. (2021). MPT-embedding: An unsupervised representation learning of code for software defect prediction. Journal of Software: Evolution and Process, 33(4), e2330. https://doi.org/10.1002/smr.2330

Staudemeyer, R. C., & Morris, E. R. (2019). Understanding LSTM--a tutorial into Long Short-Term Memory Recurrent Neural Networks. arXiv preprint arXiv:1909.09586.https://arxiv.org/abs/1909.09586

Tan, M., Tan, L., Dara, S., & Mayeux, C. (2015, May). Online defect prediction for imbalanced data. In 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering (Vol. 2, pp. 99-108). IEEE. https://doi.org/10.1109/ICSE.2015.139

Tantithamthavorn, C., McIntosh, S., Hassan, A. E., &Matsumoto, K. (2018). The impact of automated parameter optimization on defect prediction models. IEEE Transactions on Software Engineering, 45(7), 683-711. https://ieeexplore.ieee.org/abstract/document/8263202/

Tian, J., & Tian, Y. (2020, August). A Model Based on Program Slice and Deep Learning for Software Defect Prediction. In 2020 29th International Conference on Computer Communications and Networks (ICCCN) (pp. 1-6). IEEE. https://doi.org/10.1109/ICCCN49398.2020.9209658

Tong, H., Liu, B., & Wang, S. (2018). Software defect prediction using stacked denoising autoencoders and two-stage ensemble learning. Information and Software Technology, 96, 94-111. https://doi.org/10.1016/j.infsof.2017.11.008

Torres-Carrión, P. V., González-González, C. S., Aciar, S., & Rodríguez-Morales, G. (2018, April). Methodology for systematic literature review applied to engineering and education. In 2018 IEEE Global Engineering Education Conference (EDUCON) (pp. 1364-1373). IEEE. https://doi.org/10.1109/EDUCON.2018.8363388

VOSviewer. (2020). VOSviewer scientific landscapes. https://www.vosviewer.com/

Wahono, R. S. (2015). A systematic literature review of software defect prediction. Journal of Software Engineering, 1(1), 1-16. ISSN-10: 2356-3974.

Wang, S., Liu, T., & Tan, L. (2016, May). Automatically learning semantic features for defect prediction. In 2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE) (pp. 297-308). IEEE. https://doi.org/10.1145/2884781.2884804

Wang, S., Liu, T., Nam, J., & Tan, L. (2018). Deep semantic feature learning for software defect prediction. IEEE Transactions on Software Engineering, 46(12), 1267-1293.

Wang, Z., & Lu, L. (2020). A Semantic Convolutional Auto-Encoder Model for Software Defect Prediction, Reference Number, pp. 1-6, 2020. https://doi.org/10.18293/SEKE2020-036

Wen, M., Wu, R., & Cheung, S. C. (2018). How well do change sequences predict defects? sequence learning from software changes. IEEE Transactions on Software Engineering, 46(11), 1155-1175.

Yang, X., Lo, D., Xia, X., Zhang, Y., & Sun, J. (2015, August). Deep learning for just-in-time defect prediction. In 2015 IEEE International Conference on Software Quality, Reliability and Security (pp. 17-26). IEEE. https://doi.org/10.1109/QRS.2015.14

Yuan, X., Gu, Y., & Wang, Y. (2020). Supervised deep belief network for quality prediction in industrial processes. IEEE Transactions on Instrumentation and Measurement, 70, 1-11. https://doi.org/10.1109/TIM.2020.3035464

Zhang, N., Zhu, K., Ying, S., & Wang, X. (2020). Software defect prediction based on stacked contractive autoencoder and multi-objective optimization. Computers, Materials and Continua, 65(1), 279-308. https://doi.org/10.32604/cmc.2020.011001

Zhang, Q., & Wu, B. (2020, June). Software Defect Prediction via Transformer. In 2020 IEEE 4th Information Technology, Networking, Electronic and Automation Control Conference (ITNEC) (Vol. 1, pp. 874-879). IEEE. https://doi.org/10.1109/ITNEC48623.2020.9084745

Zhang, X., Ben, K., & Zeng, J. (2018a, July). Cross-entropy: A new metric for software defect prediction. In 2018 IEEE International Conference on Software Quality, Reliability and Security (QRS) (pp. 111-122). IEEE. https://doi.org/10.1109/QRS.2018.00025

Zhang, Y., Lo, D., Xia, X., & Sun, J. (2018b). Combined classifier for cross-project defect prediction: an extended empirical study. Frontiers of Computer Science, 12(2), 280-296. https://doi.org/10.1007/s11704-017-6015-y

Zhou, M., Chen, J., Hu, H., Yu, J., Li, Z., & Hu, H. (2019, December). Deeptle: Learning code-level features to predict code performance before it runs. In 2019 26th Asia-Pacific Software Engineering Conference (APSEC) (pp. 252-259). IEEE. https://doi.org/10.1109/APSEC48747.2019.00042

Zhou, X., & Lu, L. (2020, December). Defect Prediction via LSTM Based on Sequence and Tree Structure. In 2020 IEEE 20th International Conference on Software Quality, Reliability and Security (QRS) (pp. 366-373). IEEE. https://doi.org/10.1109/QRS51102.2020.00055

Zhu, K., Zhang, N., Ying, S., & Zhu, D. (2020A). Within-project and cross-project just-in-time defect prediction based on denoising autoencoder and convolutional neural network. IET Software, 14(3), 185-195. https://doi.org/10.1049/iet-sen.2019.0278

Zhu, K., Zhang, N., Zhang, Q., Ying, S., & Wang, X. (2020B). Software defect prediction based on non-linear manifold learning and hybrid deep learning techniques. Computers, Materials and Continua, 65(2), 1467-1486. https://doi.org/10.32604/cmc.2020.011415

Zhu, Y., Yin, D., Gan, Y., Rui, L., & Xia, G. (2019, May). Software Defect Prediction Model Based on Stacked Denoising Auto-Encoder. In International Conference on Artificial Intelligence for Communications and Networks (pp. 18-27). Springer, Cham. https://doi.org/10.1007/978-3-030-22971-9_2