Hybrid Software Defect Prediction Based on LSTM (Long Short Term Memory) and Word Embedding

Conference Paper · October 2021

DOI: 10.1109/ICON-SONICS53103.2021.9617182

Rizal Broer Bahaweres
Syarif Hidayatullah State Islamic University Jakarta
62 PUBLICATIONS **159** CITATIONS
SEE PROFILE

Detia Jumral
Syarif Hidayatullah State Islamic University Jakarta
1 PUBLICATION **0** CITATIONS
SEE PROFILE

Irman Hermadi
Bogor Agricultural University
85 PUBLICATIONS **390** CITATIONS
SEE PROFILE

Arif Imam Suroso
Bogor Agricultural University
72 PUBLICATIONS **139** CITATIONS
SEE PROFILE

Hybrid Software Defect Prediction Based on LSTM (Long Short Term Memory) and Word Embedding

Rizal Broer Bahaweres[1] , Detia Jumral[2], Irman Hermadi[3], Arif Imam Suroso[4], Yandra Arkeman[5],
[1,2]Department of Informatics, Syarif Hidayatullah State Islamic University, Jakarta, Indonesia
[1,3]Computer Science Dept, IPB University, Bogor, Indonesia
[4]School of Business, IPB University, Bogor Indonesia
[5]Department of Agro-Industrial Tech., IPB University, Bogor, Indonesia
rizalbroer@computer.org, jumral.detia17@mhs.uinjkt.ac.id, irmanhermadi@apps.ipb.ac.id, Arifimamsuroso@apps.ipb.ac.id, yandra.arkeman@gmail.co

Abstract— **Defects often occur in software so it can cause various problems for users. The running system may have defects, although they are not immediately apparent. Defects can be UI or display defects or defects in logic code. In predicting software defects several methods have been developed focusing on software metrics (Line of Code, Cyclomatic Complexity, etc.). But in capturing program syntax and semantics and the ability to build accurate predictive models, software metrics often fail. We propose a method that combines deep learning methods and word embedding to predict software defects. We will map tokens using an abstract syntax tree from the source code and then build an LSTM network to predict software defects. The dataset used is taken from an open source Java project, namely the Apache Project Repository. The evaluation results show that the LSTM method combined with word embedding can get accuracy 98%, precision 89%, recall 92% and f1-score 90%. This is greater than other software defect prediction methods.**

Index Terms—Software Defect Prediction, Word Embedding, Long Short-Term Memory.

I. INTRODUCTION

Defects are a fundamental property of a system. They come from design or manufacturing, as well as external problems. A system that is running well at this time may also have defects that are not realized or are not so important at this time. A software flaw is a programming error that causes things differently than expected. Software quality is one of the researches in the field of software engineering which has a significant role in the creation of high-quality software systems. Prediction of software defects caused by various deviations from the specification process or something that might cause operational failures that have been the topic of research for more than 30 years [1].

Software defect prediction is a process that can determine the possible flawed parts of a software system [2]. Defect prediction strategies typically use a model to expect whether or not a brand new module incorporates defects. Defect prediction strategies typically use a version to expect whether or not a brand new module incorporates defects. Software program metrics designed manually to assemble classifiers have been applied by several previous studies. Some of the methods that are often used are Decision Tree (DT), Support Vector Machine (SVM), Neural Network (NN), Naive Bayes (NB) [3]. The effectiveness of defect prediction depends on the data pre-processing and ensemble development, modeling technique and parameter optimization. [4].

The failure rate of software projects in the world reaches more than 40% according to statistical data. In America, there were 42% failures recorded by the Standish Group, while the General Accounting Office found that software development failures reached 53% in 2012. The total distribution of defect prediction methods is 77.46%. Research studies related to classification methods, research focusing on estimation methods is 14.08%, and research related to clustering methods and associations is 1.41%. research studies use public data sets with 64.79% and research studies use private data sets with 35.21% [5]. This research is part of the Data Research and Software Engineering research group. Software Defect Prediction research began around the year 2017 [6], 2018 [7], 2020 [8] . We tried to update by adding a few things from the papers.

In software systems, defects often occur and can cause various problems for software users. Several methods were developed to predict the location of the most likely defects quickly and precisely. However his approach does not quite discover the syntax and varying degrees of semantics of source code, capabilities that are much needed to create accurate predictive models. Software defect prediction can assist developers in discoverability defects and decrease maintenance costs. The conventional method generally makes use of software program metrics (Code Lines, Cyclomatic Complexity, etc). In capturing software syntax semantic information, software metrics often fail. Seen when using a framework that combines semantic techniques and in-depth study for defect prediction. According to the Big Indonesian Dictionary, semantics is the science of the meaning of words and sentences, knowledge of the intricacies and shifts in the meaning of words; the part of the structure of language that relates to the meaning of an expression or the structure of the meaning of a speech. One of the semantics used is ontology. This ontology is a philosophical science by studying what exists. The purpose of this ontology is to guess something based on the neighbor word. And the predictor area of this ontology is more accurate. To build a Short-Term Memory (LSTM) network using a vector chain and its labels (disabled or non-defective). The semantic information of this system and perform defect prediction is analyzed mechanically by the LSTM model. From the existing problems, the author will examine "Software Defect Prediction with Semantic Relations and LSTM (Long Short-Term Memory)". For the LSTM method, we carried out 2 scenarios, stateless LSTM and stateful LSTM. We want to prove that it is true that LSTM can actually store memory in the long term. And we did a comparison with the RNN (Recurrent Neural Network) method.

In this study the formulation of the problem can be determined as follows:

1. How do the results of accuracy, recall, precision, and f1 score on LSTM in scenario I and LSTM scenario II?
2. How does the semantic LSTM and RNN compare for software defect prediction?

The scope of problem in this study are as following:

1. The Long Short-Term Memory (LSTM) algorithm is used to build the prediction model.
2. The dataset used is taken from an open-source Java project, namely the Apache Project Repository.
3. Using the Python programming language and Scikit-Learn, TensorFlow.

II. Literature review

A. Related Search

Research with [9] using dataset from Samsung and public dataset from PROMISE repository. They develop predictive models that can study the source code. They use an abstract syntax tree representation that is compatible with the LSTM-tree modeling. They use the DTL-DP (Deep Transfer Learning) method which proposes an end-to-end framework that can predict programs without feature extraction. They visualize the program as an image and then apply a self-attention mechanism to extract the image features, which then inserts the image file into a pre-trained deep learning model. Their research used a dataset from the PROMISE repository [10]. Furthermore [11] using a semantic algorithm based on DBN (Deep Belief Network) and performing evaluation metrics. They use publicly available data from the PROMISE repository. The data set was used to compare their approach with existing defect prediction models in the same dataset. Then in 2017 [12], in this study they used the CNN (Convolutional Neural Network) algorithm and the dataset used was the PROMISE repository. They extract the token vector which is then encoded into a numeric vector in an abstract syntax tree and then uses CNN to automate the source code with structural semantics and take advantage of word insertion and merging.

B. Word Embedding

The characteristic learning approach in textual content processing or Natural Language Processing (NLP) to build a low-dimensional word vector illustration from a group of textual content is called word embedding. The primary benefit of word embedding is that it permits to provide an extra expressive and efficient illustration with the aid of using keeping the contextual similarity of phrases and with the aid of using building low-dimensional vectors [13]. Word embeddings are one of the most popular strategies in natural language processing. Word embedding can enhance function overall performance and the additional use of classifiers primarily based totally on LSTM [14].

The process by which words are converted into vectors of real numbers is called word embedding. The most basic way to convert words to numbers is through the encoding method. Each word will have a unique dimension. Mapped into a vector space, words used in similar contexts will be close together whereas words that will be far apart in a vector space. Data preparation such as data cleaning, data outliers and feature selection or often called feature engineering. The feature engineering strategy in textual data that is often used is known as the Bag of Words model. The Continuous Bag-of-Word (CBOW) model and the Skip-Gram model are architectural models of word2Vec. Word2Vec predicts words based on word context using one of its architectural models. The CBOW model uses context to predict the target word. CBOW also has less training time and has good accuracy for frequent words frequent [15].

C. LSTM

Long Short-Term Memory (LSTM) is a type of Neural Network architecture. LSTM has the ability to remember long-term information. In Figure 1 an LSTM has four layers in the loop [16].

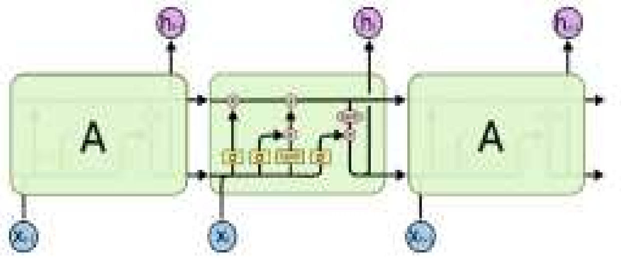

Figure 1 Looping with four layers on LSTM

Each Long Short-Term Memory cell, an LSTM cell, has an internal state. For every input, the information saved withinside the internal state is up to date with the aid of using including and subtracting information. The LSTM model was created to overcome the gradient problem that often occurs in most iterative neural network models [17]. According to Hochreiter & Schmidhuber in 1997, the equations that exist in the LSTM method can be described in the equation below:

$$ft = \sigma\,(Wf\ .\ [ht{-}1, xt] + bf)$$
$$it = \sigma\,(Wi\ .\ [ht{-}1, xt] + bi)$$
$$\tilde{Ct} = \tanh\,(Wc.\ [ht{-}1, xt] + bc)$$
$$Ct = ft * Ct{-}1 + it * \tilde{Ct}$$
$$ot = \sigma\,(Wo\ .\ [ht{-}1, xt] + bo)$$
$$ht = ot * \tanh Ct$$

D. Confusion Matrix

Confusion Matrix is a machine learning method that has facts approximately the actual state of the data and the prediction consequences of a class that has been accomplished the usage of the version or classification method used. Classification performance is evaluated using data in a matrix containing actual data, a prediction result from the classification method used. Some classification performance calculations may be defined from the confusion matrix [18].

- Accuracy

  The percentage of the total number of correct predictions in the classification process is called accuracy [19].

$$Accuracy = \frac{TP + TN}{n}$$

- Recall

  The percentage of positive data that is predicted to be a positive value is called recall [20].

$$Recall = \frac{TP}{TP + FN}$$

- Precision

  The ratio of the correct positive prediction to the overall positive predictive result is called precision.

$$Precission = \frac{TP}{TP + FP}$$

- F1-Score

  The weighted comparison of the average precision and recall is called the F1-Score.

$$F1\ Score = \frac{2\ x\ Precision\ x\ Recall}{Precision + Recall}$$

III. RESEARCH METHODOLOGY

The methodology that will be used in this study is shown in Figure 2. Collecting datasets drawn from open-source Java projects is the first stage. We then carry out the preprocessing process by tokenizing and filtering. Then divide the dataset into training datasets and testing the datasets followed by word embedding. We will use the data that has been word embedded to find accuracy, recall, precision, and also F1-score using semantic relations and LSTM (Long Short Term-Memory).

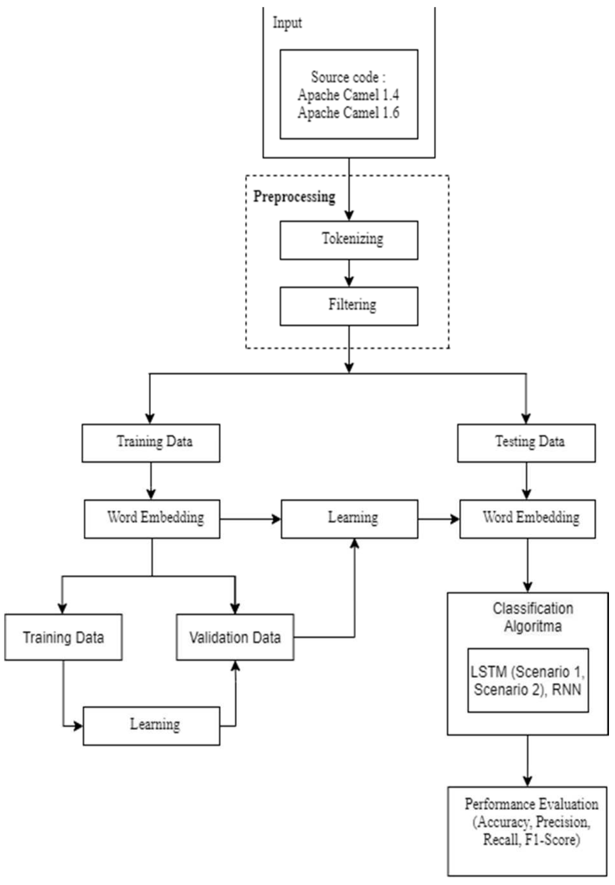

Figure 2 Methodology

A. Dataset

In this study, the data was taken from the Java open source project, namely the Apache Project Repository. Using Apache Camel versions 1.4 and 1.6 where there are 1839 total number of code files. These code are then divided into defect and non-defect labels. In camel 1.4 there were 145 defect and 683 non-defects. While in camel 1.6 there were 186 defect and 725 non-defects. Then extract the sequence of tokens for the project's Java files. In table 1, many code are used as datasets.

TABLE I
DETAILS OF NUMBER OF DATASET BY DEFECT AND NON-DEFECT

	Defect	***Non-Defect***
Camel 1.4	146	683
Camel 1.6	186	725
Total	331	1408

B. Preprocessing

The input will then be processed at the preprocessing stage. This stage is carried out so that the data held is clean so that it can be processed using the LSTM model. At this stage there are several processes that are passed, namely tokenizing and filtering. In the tokenizing stage, there is a process of cutting words into one syllable. Tokenizing is a process of parsing text content into words, terms, symbols, or elements that make up a text. These descriptions are called tokens. In this process, characters such as spaces, periods (.), commas (,), and other characters that are used as separators will be removed. This token list will be used as input for further processing of the text representation. In filtering there is a process of removing punctuation marks such as (!”#$%&'()*+,-./:;<=>?@[\]^_`{|}~) which will be replaced with space characters. The removal of punctuation marks is done because the punctuation training process will be ignored so that the training process will be simpler.

C. Word Embedding

Word Embedding process by looking for relationships between words. All words in the training data were trained using word2vec. Each word will be created based on the word order in the word library. After being trained, the training data will be made into a matrix, where each word will be searched for its relationship value according to the dimensions used. Using the dimensions of 5000 words in the word embedding matrix. This matrix will be used as a weight for the Embedding layer used by the LSTM model to train the dataset.

D. Long Short Term Memory Architecture

Network architecture is formed to produce optimal accuracy. The training model can have various number of layers, but in this study it consists of four layers namely Embedding layer, LSTM layer, Dense layer with various input features. The overall dataset is divided into three categories: training data, validation data, and test data. In addition, the Adam optimization algorithm and binary cross entropy loss are used for the optimizer and loss functions, respectively. For sigmoid activation and ReLU functions are used. In training the model not only optimization or loss function but also with a significant number of epochs, where when the entire dataset is passed through the neural network forward (forward) and backward (backpropagation) it is only called one epoch. For LSTM the hyperparameter uses several layers. embedding layer with optimization of 5000, LSTM layer 128, Dense layer 128, Dropout layer 128, and using epoch 10 times.

E. Classification Performance Calculation

After building the LSTM and RNN network architecture, the next step is to calculate the level of classification performance such as accuracy, precision, recall, and f1-score.

IV. RESULT AND DISCUSSION

A. Prediction Result

For the LSTM method, we carried out 2 scenarios, namely stateless LSTM and stateful LSTM. And do a comparison with the RNN method.

1) Scenario I : LSTM Stateless

In scenario I, we run the runtime 3 times, but before running we restart the runtime before doing the next runtime. In table II, it can be seen the results of 3 experiments with the average results on the camel-1.4 dataset. Table III is the result of doing 3 experiments and the average result of the 3 experiments using the camel-1.6 dataset.

TABLE II
CLASSIFICATION PERFORMANCE LSTM SCENARIO I DATASET I

LSTM Camel-1.4				
	Accuracy	Precision	Recall	F1-S
Experimen 1	0,989	0,863	0,9264	0,8936
Experimen 2	0,9835	0,8684	0,9496	0,9073
Experimen 3	0,9862	0,8943	0,9071	0,9007
AVERAGE	0,9862	0,8752	0,9277	0,9005

TABLE III
CLASSIFICATION PERFORMANCE LSTM SCENARIO I DATASET II

LSTM Camel-1.6				
	Accuracy	Precision	Recall	F1-S
Experimen 1	0,9865	0,8322	0,8958	0,8628
Experimen 2	0,9910	0,8129	0,9064	0,8571
Experimen 3	0,9910	0,8618	0,8911	0,8762
AVERAGE	0,9895	0,8356	0,8978	0,8654

2) Scenario II : LSTM Stateful

In scenario II, we run the runtime 3 times in sequence without restarting it first. In table IV, it can be seen the results of 3 experiments with the average results using the camel-1.4 dataset. Table V is the result of doing 3 experiments and getting the average results of the 3 experiments using the camel-1.6 dataset.

TABLE III
CLASSIFICATION PERFORMANCE LSTM SCENARIO II DATASET I

LSTM Camel-1.4				
	Accuracy	Precision	Recall	F1-S
Experimen 1	0,989	0,8857	0,9117	0,8985
Experimen 2	0,9862	0,8671	0,9185	0,892
Experimen 3	0,989	0,8675	0,9562	0,9097
AVERAGE	0,9880	0,8734	0,9288	0,9007

TABLE III
CLASSIFICATION PERFORMANCE LSTM SCENARIO II DATASET II

LSTM Camel-1.6				
	Accuracy	Precision	Recall	F1-S
Experimen 1	0,994	0,9205	0,9025	0,9114
Experimen 2	0,985	0,8843	0,8609	0,8724
Experimen 3	0,9865	0,8941	0,8941	0,894
AVERAGE	0,9885	0,8996	0,8858	0,8926

TABLE IV
CLASSIFICATION PERFORMANCE REPORT

	Dataset Name	**Accuracy**	**Precision**	**Recall**	**F1-S**
LSTM Scenario I	Camel-1.4	0,9862	**0,8752**	0,9277	0,9005
	Camel-1.6	**0,9895**	0,8356	**0,8978**	0,8654
LSTM Scenario II	Camel-1.4	**0,9880**	0,8734	**0,9288**	**0,9007**
	Camel-1.6	0,9885	**0,8996**	0,8858	**0,8926**
RNN	Camel-1.4	0,8512	0,8442	0,7573	0,7984
	Camel-1.6	0,8848	0,8357	0,8068	0,8210

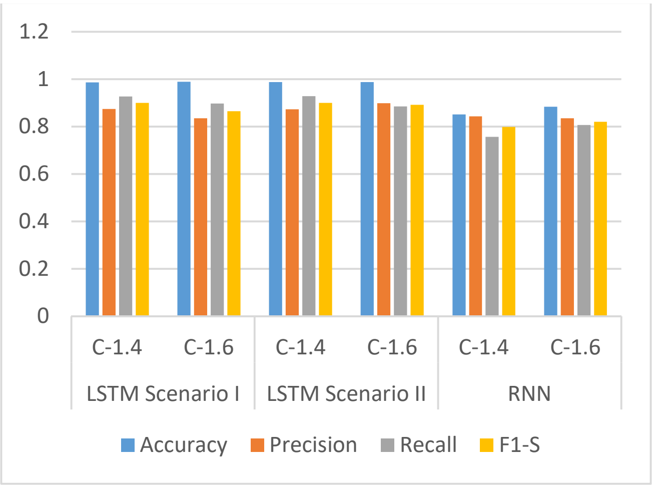

Figure 3 Classification Performance

B. Result

In the LSTM network, we do 2 scenarios. Given the LSTM can add memory cells that can store information for a long period of time. So, we tried scenario I by doing 3 times

Experiment but before starting Experiment 2 we restarted the runtime first. Likewise when doing the third Experiment. From the results of these 3 experiments, we will calculate the average value. Furthermore, for scenario II, we did 3 experiments as well but they were carried out without restarting before doing the second experiment (sequentially). From these 3 experiments we also calculate the average value. We also conducted trials using the Recurrent Neural Network (RNN).

The results of scenario I obtained: in table III with the camel-1.4 dataset, it can be seen that the average obtained after doing 3 times the experiment with restart is 98% accuracy score with 87% precision value, 92% recall value and f1-score value. as much as 90%. While table IV on the camel-1.6 dataset can be seen that the average value obtained after doing 3 experiments with restart is 98% accuracy value with 83% precision value, 89% recall value and 86% f1-score value.

Scenario II: table V dataset camel-1.4 can be seen that the average value obtained after doing 3 experiments consecutively without restarting is 98% accuracy value with 87% precision value, 92% recall value and 90 f1-score value. %. For the camel-1.6 dataset in table VI, it can be seen that the average value obtained after doing 3 consecutive experiments without restarting is 98% accuracy with 89% precision value, 88% recall value and 89% f1-score value.

In the RNN network, we get accuracy, precision, recall, and f1-score values in dataset 1, respectively, 85%, 84%, 75% and 79%. In dataset 2, 88% accuracy is obtained, 83% precision, 80% recall, and 82% f1-score.

It can be seen that prediction using the LSTM network can prove to be superior to the RNN network. For LSTM with restart and successive LSTM, it can be seen that the camel-1.4 dataset for accuracy, recall, and also f1-score LSTM sequentially obtained a higher score. While precision is superior to using LSTM with restart. For the camel-1.6 dataset on precision and also f1-score the results obtained are superior to sequential LSTMs. while for accuracy and recall the score results are superior to LSTM with restart. However, the prediction results with this LSTM network in both scenario 1 and scenario 2 do not have a significant difference.

V. CONCLUSION AND FUTURE WORK

In our prediction model, we take the AST as input for the general representation for the source code and predict whether the file is corrupted or not. To capture the long-term dependencies that often exist between code elements we use a prediction system on a deep learning LSTM (Long Short-Term Memory) network. We evaluate with a dataset from an opensource Java project, namely from the Apache Project Repository. From the results of this study, it was concluded that the results of training using the LSTM network were both done by restarting and sequentially got higher accuracy, precision, recall, and f1-score results than using the RNN algorithm. The highest accuracy value is obtained by 98% using LSTM then precision is obtained at 89%, recall is 92% and f1-score is 90%. We did a comparison between LSTM 3 Experiments with restart and also LSTM 3 Experiments sequentially. For each dataset used, there are some differences in the results of the confusion matrix, but the differences are not significant. Meanwhile, when comparing with the RNN algorithm, the results obtained on the LSTM are superior. This shows that our approach can be applied in practice.

For further research, it is expected to carry out dataset processing methods so as to get a cleaner and more precise dataset so as to increase accuracy for the better. And also to use other word embedding methods such as TF-IDF, Skip-Gram and also other word embedding methods. And can do with more diverse datasets.

REFERENCES

[1] R. Setya Perdana and U. L. Yuhana, "Perdana, Prediksi Code Defect Perangkat Lunak Dengan Metode Association Rule Mining dan Cumulative Support Thresholds 113 Prediksi Code Defect Perangkat Lunak Dengan Metode Association Rule Mining dan Cumulative Support Thresholds."

[2] B. Kitchenham, B. Kitchenham, and S. Charters, "Guidelines for performing Systematic Literature Reviews in Software Engineering," 2007, Accessed: May 06, 2020. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.117.47 1.

[3] H. Liang, Y. U. E. Yu, L. I. N. Jiang, and Z. Xie, "Seml : A Semantic LSTM Model for Software Defect Prediction," *IEEE Access*, vol. 7, pp. 83812–83824, 2019, doi: 10.1109/ACCESS.2019.2925313.

[4] K. Tanaka, A. Monden, and Z. Yucel, "Prediction of Software Defects Using Automated Machine Learning," in *Proceedings - 20th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing, SNPD 2019*, Jul. 2019, pp. 490–494, doi: 10.1109/SNPD.2019.8935839.

[5] R. S. Wahono, "A Systematic Literature Review of Software Defect Prediction : Research Trends , Datasets , Methods and Frameworks," vol. 1, no. 1, 2015.

[6] R. B. Bahaweres, K. Zawawi, D. Khairani, and N. Hakiem, "Analysis of statement branch and loop coverage in software testing with genetic algorithm," *Int. Conf. Electr. Eng. Comput. Sci. Informatics*, vol. 2017-December, no. September, pp. 19–21, 2017, doi: 10.1109/EECSI.2017.8239088.

[7] P. A. Habibi, V. Amrizal, and R. B. Bahaweres, "Cross-Project Defect Prediction for Web Application Using Naive Bayes (Case Study: Petstore Web Application)," *2018 Int. Work. Big Data Inf. Secur. IWBIS 2018*, pp. 13–18, 2018, doi: 10.1109/IWBIS.2018.8471701.

[8] R. B. Bahaweres, F. Agustian, I. Hermadi, A. I. Suroso, and Y. Arkeman, "Software Defect Prediction Using Neural Network Based SMOTE," in *2020 7th International Conference on Electrical Engineering, Computer Sciences and Informatics (EECSI)*, Oct. 2020, no. October, pp. 71–76, doi: 10.23919/EECSI50503.2020.9251874.

[9] H. K. Dam *et al.*, "A deep tree-based model for software defect prediction," Feb. 2018, [Online]. Available: http://arxiv.org/abs/1802.00921.

[10] J. Chen *et al.*, "Software visualization and deep transfer learning for effective software defect prediction," pp. 578–589, 2020, doi: 10.1145/3377811.3380389.

[11] S. Wang, T. Liu, J. Nam, and L. Tan, "Deep Semantic Feature

Learning for Software Defect Prediction," *IEEE Trans. Softw. Eng.*, vol. 5589, no. c, pp. 1–26, 2018, doi: 10.1109/TSE.2018.2877612.

[12] J. Li, P. He, J. Zhu, and M. R. Lyu, "Software defect prediction via convolutional neural network," *Proc. - 2017 IEEE Int. Conf. Softw. Qual. Reliab. Secur. QRS 2017*, pp. 318–328, 2017, doi: 10.1109/QRS.2017.42.

[13] M. Naili, A. H. Chaibi, and H. H. Ben Ghezala, "Comparative study of word embedding methods in topic segmentation," *Procedia Comput. Sci.*, vol. 112, pp. 340–349, 2017, doi: 10.1016/j.procs.2017.08.009.

[14] A. Jimeno Yepes, "Word embeddings and recurrent neural networks based on Long-Short Term Memory nodes in supervised biomedical word sense disambiguation," *J. Biomed. Inform.*, vol. 73, pp. 137–147, 2017, doi: 10.1016/j.jbi.2017.08.001.

[15] A. Nurdin, B. Anggo Seno Aji, A. Bustamin, and Z. Abidin, "Perbandingan Kinerja Word Embedding Word2Vec, Glove, Dan Fasttext Pada Klasifikasi Teks," *J. Tekno Kompak*, vol. 14, no. 2, p. 74, 2020, doi: 10.33365/jtk.v14i2.732.

[16] C. Olah, "Understanding LSTM Networks [Blog]," *Web Page*, pp. 1–13, 2015, [Online]. Available: http://colah.github.io/posts/2015-08-Understanding-LSTMs/.

[17] J. Patterson and A. Gibson, *Deep Learning: A Practitioner's Approach*. United States of America: O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472, 2017.

[18] P. Kumudha and R. Venkatesan, "Cost-Sensitive Radial Basis Function Neural Network Classifier for Software Defect Prediction," *Sci. World J.*, vol. 2016, pp. 1–20, 2016, doi: 10.1155/2016/2401496.

[19] X. Deng, Q. Liu, Y. Deng, and S. Mahadevan, "An improved method to construct basic probability assignment based on the confusion matrix for classification problem," *Inf. Sci. (Ny).*, vol. 340–341, pp. 250–261, 2016, doi: 10.1016/j.ins.2016.01.033.

[20] Saraswathi and Sheela, "Lung Image Segmentation Using K-Means Clustering Algorithm with Novel Distance Metric," *Int. J. Recent Trends Eng. Res.*, vol. 2, no. 12, pp. 236–245, 2016.