sensors

MDPI

Review

Machine Learning-Based Software Defect Prediction for Mobile Applications: A Systematic Literature Review

Manzura Jorayeva [1], Akhan Akbulut [1], Cagatay Catal [2] and Alok Mishra [3,*]

[1] Department of Computer Engineering, Istanbul Kültür University, Istanbul 34158, Turkey; 1800004575@stu.iku.edu.tr (M.J.); a.akbulut@iku.edu.tr (A.A.)
[2] Department of Computer Science and Engineering, Qatar University, Doha 2713, Qatar; ccatal@qu.edu.qa
[3] Informatics and Digitalization Group, Faculty of Logistics, Molde University College-Specialized University in Logistics, 6410 Molde, Norway
* Correspondence: alok.mishra@himolde.no

Abstract: Software defect prediction studies aim to predict defect-prone components before the testing stage of the software development process. The main benefit of these prediction models is that more testing resources can be allocated to fault-prone modules effectively. While a few software defect prediction models have been developed for mobile applications, a systematic overview of these studies is still missing. Therefore, we carried out a Systematic Literature Review (SLR) study to evaluate how machine learning has been applied to predict faults in mobile applications. This study defined nine research questions, and 47 relevant studies were selected from scientific databases to respond to these research questions. Results show that most studies focused on Android applications (i.e., 48%), supervised machine learning has been applied in most studies (i.e., 92%), and object-oriented metrics were mainly preferred. The top five most preferred machine learning algorithms are Naïve Bayes, Support Vector Machines, Logistic Regression, Artificial Neural Networks, and Decision Trees. Researchers mostly preferred Object-Oriented metrics. Only a few studies applied deep learning algorithms including Long Short-Term Memory (LSTM), Deep Belief Networks (DBN), and Deep Neural Networks (DNN). This is the first study that systematically reviews software defect prediction research focused on mobile applications. It will pave the way for further research in mobile software fault prediction and help both researchers and practitioners in this field.

Keywords: software defect prediction; software fault prediction; mobile application; review; systematic literature review; deep learning; machine learning

Citation: Jorayeva, M.; Akbulut, A.; Catal, C.; Mishra, A. Machine Learning-Based Software Defect Prediction for Mobile Applications: A Systematic Literature Review. *Sensors* **2022**, 22, 2551. https://doi.org/10.3390/s22072551

Academic Editors: Ansar Yasar, Nafaa Jabeur, Michele Melchiori and Francesco Mercaldo

Received: 17 January 2022
Accepted: 24 March 2022
Published: 26 March 2022

1. Introduction

In recent times, mobile applications play an undeniably significant role in many aspects of our lives. The new generation is growing up with new technology such as mobile phones, tablets, and laptops. We are connected by smartphones and use them for many different purposes in daily life frequently, and as such, mobile applications are becoming more and more crucial in our lives. Nowadays, we download mobile applications to access digital information, play games, learn languages, and communicate with each other. Many applications are available with free and paid versions. Compared to desktop applications, mostly mobile applications have fewer bugs/defects. Therefore, it is crucial to predict the faults before they are deployed to the mobile markets such as the Google Play Store and the iOS App Store. Currently, we download mobile applications from digital markets such as the App Store and Android Market to access information, play games, learn languages, and communicate with others. Not only are the complexities of these applications increasing dramatically, but also so are the user expectations. Many people are progressively affected by mobile application downloads, and users do not waste time using the mobile app once they observe a functional or non-functional problem [1]. Thus, it is a threat for mobile application developers and businesses to deploy software even with some minor bugs.

These bugs affect the effectiveness of mobile apps and can cause unpredictable crashes [2]. When users download these problematic versions of applications, they may encounter serious problems [3]. Diagnosing mobile applications crashes can give a chance to improve faults before new releases [4].

A defect or fault can be an internal failure of the application and cause the system to shut down [5]. Automatic tests can identify 35% to 60% of faults, and automatic tools developed by artificial neural networks can predict 70% of faults [6]. The most used approach for software fault prediction is to analyze a set of software metrics together with labeled data with respect to different software modules and then apply machine learning (ML) methods on such datasets [7]. The aim of this prediction task is to release applications without bugs. From the 1990s until now, software defect prediction models were developed to detect faults before they are deployed to the field, and defective modules were identified before system tests by using these prediction models. Software defect prediction approaches use past software metrics and defect data to predict defective components for the new software versions. In this study, we analyzed 47 articles that focused on mobile defect prediction models and evaluated them using numerous aspects. We excluded studies that do not introduce any empirical results and are not directly related to the mobile defect prediction model development. This review identifies challenges, research gaps, and potential solutions in such a way that both researchers and practitioners can benefit.

In this study, we followed the Systematic Literature Review (SLR) methodology and responded to nine research questions defined at the beginning of this research. To the best of our knowledge, there is no other systematic review study that focuses on mobile defect prediction models and therefore, this paper provides critical insights into this field. The other sections are organized as follows: Section 2 provides the background and related work. Section 3 defines the research methodology. Results are presented in Section 4. Threats to validity are shown in Section 5. Finally, Section 6 provides the conclusions and future work.

2. Background and Related Work

In Section 2.1, mobile software fault prediction studies and the use of machine learning and In Section 2.2 software metrics are explained. In Section 2.3, related studies are discussed.

2.1. Mobile Fault Prediction and Machine Learning

Machine learning research aims to identify data patterns and discover interesting knowledge from a large amount of data. Since the 1990s, software defect prediction studies have been using machine learning algorithms to identify fault-prone classes. While software metrics are calculated based on the collected data from software repositories, fault data is retrieved from issue tracking systems. There are different software tools such as Understand to calculate software metrics automatically from software projects; however, automation of fault data collection is more challenging. Machine learning has been applied for both predicting the number of faults (i.e., a regression task) and categorization of modules into fault-prone and non-fault-prone classes (i.e., binary class classification). In machine learning, there are four learning types: supervised, unsupervised, semi-supervised, and reinforcement learning. In supervised learning, labeled data are needed to build the models. In unsupervised learning, hidden structures in data are discovered by detecting the feature correlations. Clustering and dimensionality reduction algorithms are considered under the unsupervised learning category. Semi-supervised learning is used when there are very limited fault data (e.g., 5–15%). The last category is reinforcement learning that uses software agents to learn the environment by using a trial-and-error basis and also, applies feedback mechanism.

2.2. Software Metrics

The motivation for monitoring and analyzing software metrics is that they are commonly used to determine the quality of software components and/or products and to

take actions to improve their quality. In the early 1990s, object-oriented metrics were proposed and used to characterize software components. Researchers such as Chidamber and Kemerer, Li, Lorenz and Kidd have defined different metrics to measure complexity and calculate the static aspects of software design. The most well-known among them are; Number of Children (NOC), Depth of Inheritance Tree (DIT), Coupling Between Objects (CBO), Weighted Methods per Class (WMC), Lack of Cohesion in Methods (LCOM), and Response for a Class (RFC), Number of Ancestor Classes (NAC), Class Method Complexity (CMC), Number of Local Methods (NLM), Number of Descendent Classes (NDC), Coupling Through Message Passing (CTM), and Coupling Through Abstract Data Type (CTA), Number of Public Methods (NPM), Number of Methods (NM), Number of Public Variables (NPV), Number of Variables per Class (NV) [8]. In addition to these code metrics, project managers also track developer productivity, process, operational, test, and customer satisfaction metrics as well.

2.3. Related Work

In this sub-section, we discuss the previously published review papers on defect prediction. Catal and Diri analyzed software defect prediction articles with respect to different software metrics, datasets, and approaches [9]. Malhotra and Jain analyzed the prior publications and published a review paper on defect prediction [10]. Malhotra reviewed publications from 1991 to 2013 that apply machine learning methods for software defect prediction [11]. Radjenovic et al. analyzed defect prediction papers published from 1991s to 2011 and reported that machine learning methods and object-oriented metrics were widely applied for fault detection in the literature [12]. Misirli et al. analyzed 38 publications using machine learning methods and presented a systematic mapping study. They reported that machine learning algorithms such as Bayesian networks were used in 70% of studies [13]. Morera et al. reviewed studies on software defect prediction from 2002 to 2014, selected 40 studies, and presented a systematic mapping study on software defect prediction. They discussed the performance of machine learning methods such as Random Forest, Naïve Bayes, Logistic Regression, and Decision Trees [14]. Ozakıncı et al. reviewed publications published between 2000 and 2016 and selected 52 publications. They investigated the aim, development, progress, advantages, and components of models and presented a systematic review [15]. Son et al. performed a systematic mapping study of software defect prediction studies using 156 articles and reported that very few studies described cross-project defect prediction [16].

In addition, several systematic literature reviews (SLR) and systematic mapping studies (SMS) have been published in the software engineering discipline so far. Najm et al. analyzed studies published until 2017, which used a Decision Trees (DT) algorithm for software development effort estimation in their SMS study. The selected publications are categorized based on publication platform, analysis model, research strategy approaches applied in organizations [17]. Alsolai et al. analyzed publications related to the software maintainability prediction and presented an SLR study. They reported that the authors used some private datasets in some papers, evaluated their models using k-fold cross-validation approaches, and applied named regression algorithms [18]. Auch et al. analyzed studies published between 2002s and 2019, focused on similarity-based analyses on software applications, and selected 136 articles. They applied inclusion and exclusion criteria to select related studies, identified the applications' similarities, and presented a systematic literature review on similarity analysis of software applications [19]. Degu analyzed 31 studies related to the Android application memory and energy performance and published an SLR paper. This study presented a review to classify the research results covering Android application memory and energy work, resource leaks, and performance testing approaches and threats [20]. Kaur and Kaur presented an SLR study on the mobile application development and testing effort estimation. They analyzed and correlated existing test evaluation methods for conventional mobile and desktop applications [21]. Del Carpio and Angarita published an SLR study to investigate the trends in software engineering processes using

deep learning. They stated that deep learning methods such as Convolutional Neural Networks (CNN), Long-Short Term Memory (LSTM), and Recurrent Neural Networks (RNN) are used for fault detection, analyzing images, demands, and diagnose errors on the monitoring stage. In addition, they identified the usage of deep learning for defect prediction, classification problems, visualization, test, and analysis software requirements [22]. Kaur presented an SLR study on code smells and quality attributes relations. They reported that various code smells could have differing results against other software quality attributes [23].

We also observed studies that evaluated the effect of data sampling for defect prediction because the datasets in software defect prediction are mostly imbalanced and therefore, data sampling algorithms are needed. For example, Kaya et al. developed defect prediction models using machine learning algorithms, data sampling approaches, and design-level metrics. They reported that data sampling approaches help to improve the performance of models. They stated that the Adaboost ensemble algorithm provides the best performance for defect prediction [24].

Additionally, some papers presented novel models based on mobile application datasets. For example, Kaur et al. analyzed process metrics to predict defects of mobile applications and performed experiments using publicly available mobile applications datasets. They focused on regression algorithms and applied process and code metrics for model development. A process metrics-based machine learning model provided the best performance according to their experiments [25]. Zhao et al. presented a deep learning-based model for just-in-time defect prediction of Android applications. They applied their proposed model on 12 applications datasets and stated that the novel Imbalanced Deep Learning (IDL) model provided the best performance among others [26].

Additionally, some studies focused on the hyper-parameter tuning for improving the performance of models. For example, Sewak et al. analyzed different types of LSTM architectures for Intrusion Detection Systems and demonstrated the benefits of hyper-parameter tuning in LSTM models [27]. Software defect prediction can be used in many of the fields of engineering described [28] and it can be used to compare Machine Learning and Statistical methods for classification fault and non-fault classes. Internet of Things (IOT) was used to automate applications for our needs. Bhana et al. reported real-time applications of defect prediction that use restored data in the cloud. Their model can be implemented in daily life using a real-time application [29]. Pandey et al. performed a model using Long Short-Term Memory for cross-project defect prediction. They experimented with 44 projects with imbalanced datasets and compared DNNAttention with unsupervised learning [30].

3. Research Methodology

This section presents the research methodology. This study followed a similar methodology proposed by [31]. We organized the systematic review process shown in Figure 1 and followed steps to reduce risk bias in the study. First, research questions were identified, and papers were retrieved from scientific databases. Study selection criteria were applied to the papers, and a subgroup was selected for the quality assessment step. Each paper was scored based on eight quality assessment questions shown. Data were extracted and synthesized then the final subgroup of studies (i.e., 47 papers) were selected to respond to the research questions. All of these 47 papers were read in full and research questions were answered. Research questions of this study are presented in Table 1.

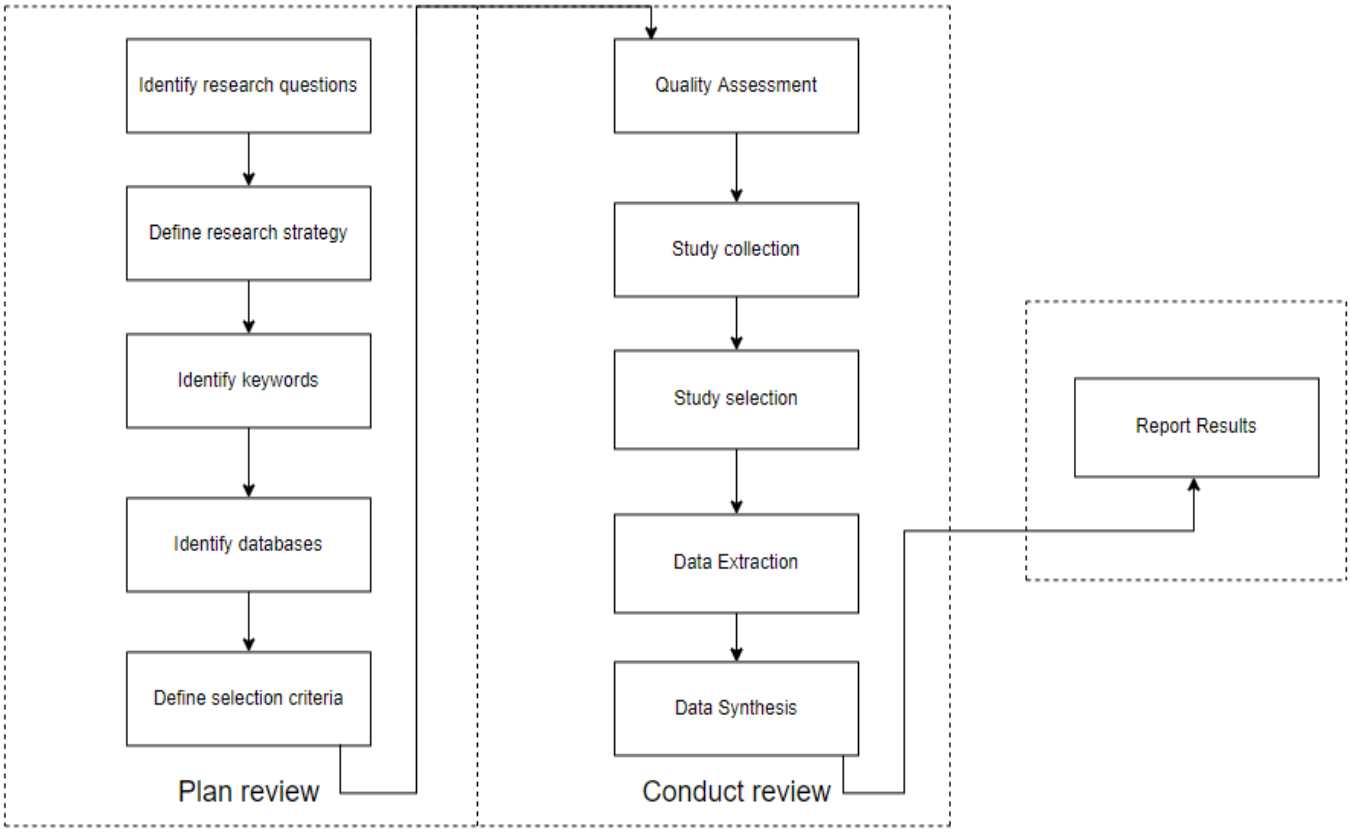

Figure 1. Systematic Literature Review process.

Table 1. Research questions in this research.

RQ	Research Questions
RQ1	Which platforms are addressed in mobile defect prediction?
RQ2	Which datasets are used in mobile defect prediction studies?
RQ3	Which machine learning types are used in mobile defect prediction studies?
RQ4	Which machine learning algorithms are applied in mobile defect prediction?
RQ5	Which evaluation metrics are used in mobile defect prediction?
RQ6	Which validation approaches were used in mobile defect prediction?
RQ7	Which software metrics were adopted in mobile defect prediction?
RQ8	Which ML algorithm works best for mobile defect prediction?
RQ9	What are the challenges and research gaps in mobile defect prediction?

The following databases were used to retrieve relevant papers: IEEE Xplore, Science Direct, ACM Digital Library, Wiley Online Library, Springer Link, and Google Scholar. The search spanned the last 10 years to identify up-to-date papers. Table 2 shows the exclusion criteria used in this study. The following search criteria were applied: ((“machine learning” OR “artificial intelligence”) AND “mobile software” AND (“fault prediction” OR “defect prediction” OR “software quality”)). Figure 2 shows the distribution of papers per database and the number of papers at each stage (i.e., after initial query, after exclusion criteria, and after quality assessment).

Table 2. Exclusion criteria [32].

ID	Exclusion Criteria
1.	The paper includes only an abstract (this criterion is not about the accessibility of the paper, we included both open access and subscription basis papers)
2.	The paper is not written in English
3.	The article is not a primary study paper
4.	The content does not provide any experimental results
5.	The study does not describe in detail how machine learning is applied

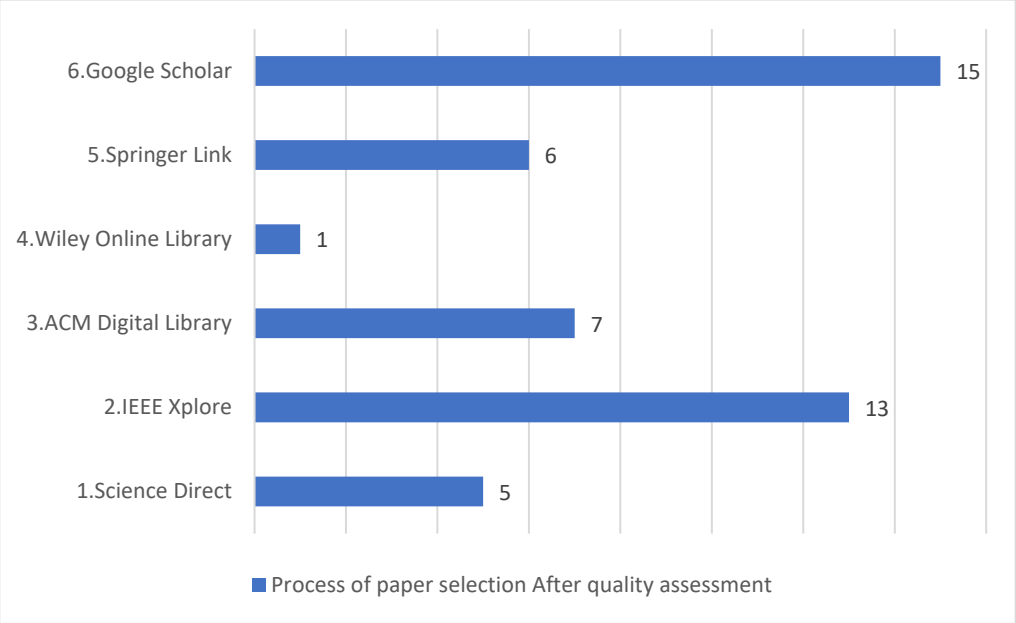

Figure 2. Distribution of the selected papers.

Figure 2 shows that most papers were retrieved from the IEEE Xplore database and Google Scholar also included a similar number of papers in the final selection.

After the exclusion criteria were applied, we graded papers for quality assessment using the approach proposed [33]. Table 3 shows quality evaluation questions. Papers with scores lower than 10 were excluded from the list. Figure 3 shows the quality distribution of papers. If the answer is "yes" for the question, the paper receives two points, the "partial" response receives one point, and "no" answer receives no points.

Table 3. Quality evaluation questions. "Yes" scores 2; "partial" scores 1; "no" scores 0.

ID	Questions
Q1	Are the aims of the study clearly declared?
Q2	Are the scope and context of the study clearly defined?
Q3	Is the proposed solution clearly explained and validated by an empirical study?
Q4	Are the variables used in the study likely to be valid and reliable?
Q5	Is the research process documented adequately?
Q6	Are all study questions answered?
Q7	Are the negative findings presented?
Q8	Are the main findings stated clearly in terms of credibility, validity, and reliability?

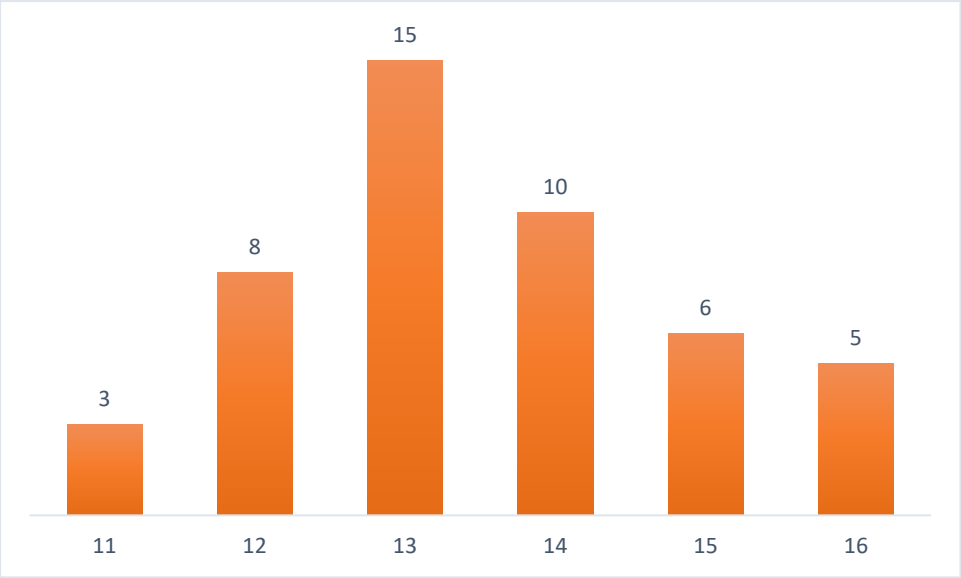

Figure 3. Quality score distribution of selected papers (x axis: paper score, y-axis: number of papers).

After quality assessment questions were applied, publications were synthesized. In Figure 4, the distribution of the selected publications per year is shown. As shown in this figure, within the last five years, more papers were published on this topic and this field is still active.

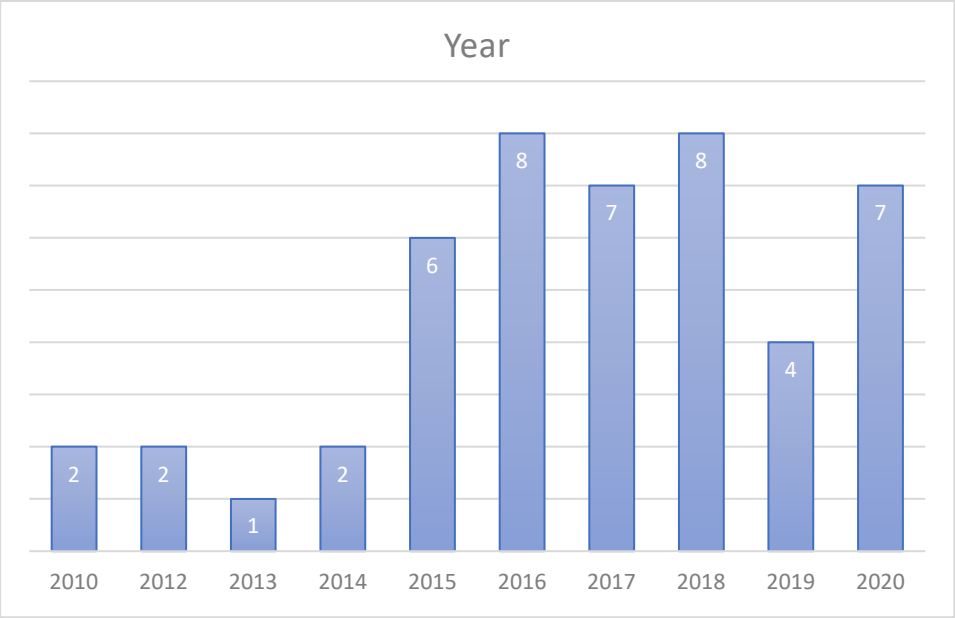

Figure 4. Selected publications per year.

Figure 5 presents the representation of the type of publications. Nearly half of the papers are journal papers, and the rest are conference proceedings. This indicates that some researchers prefer publishing this type of paper in conferences; however, a sufficient number of journal articles are evaluated in this SLR paper.

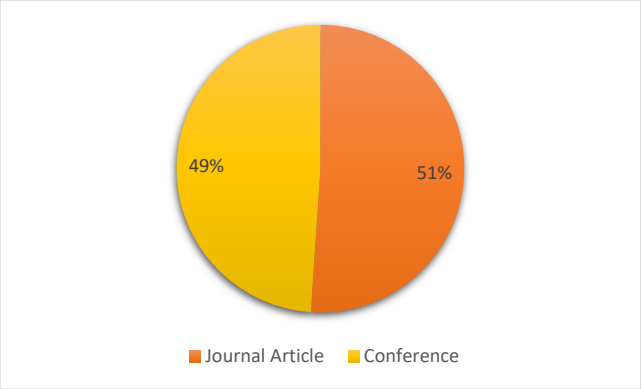

Figure 5. Distribution of type of publications.

4. Results

In this section, we explain our responses to each research question.

4.1. RQ-1: Platforms

This section provides the details of the platforms used in primary studies. As shown in Table 4, the most used platform is the Android platform, which has 21 publications. Windows Mobile platform is used only in one publication. Web applications were used in 20 publications and Mobile Applications used in five publications. This shows that most researchers prefer the Android platform for defect prediction studies. The main reason might be the open source nature of the Android platform and the applications released in this platform. There were also plenty of web applications used in these papers.

Table 4. Platforms.

Platforms	Total
Android	21
Windows Phone	1
Web Applications	20
Mobile Applications	5

4.2. RQ-2: Datasets

The datasets related to software defect prediction studies are available in repositories. Table 5 presents the repositories, datasets, and web addresses.

Figure 6 shows the distribution of repositories. As shown in the figure, most researchers preferred Github repository to host their datasets and other repositories such as SourceForge are not widely preferred.

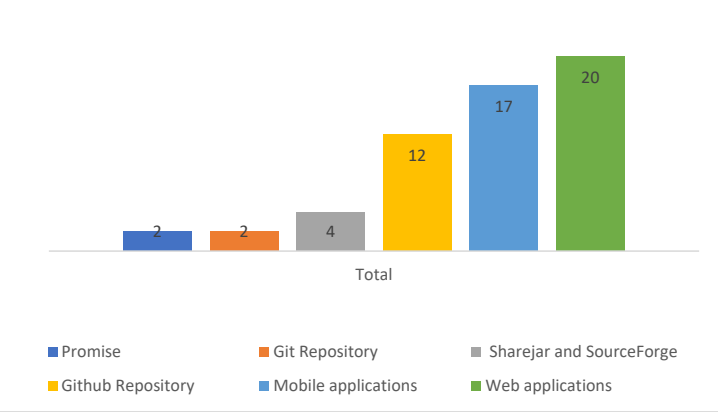

Figure 6. Repositories.

Table 5. Repositories.

Repositories	Datasets	Web Address
GIT Repository	Contact, MMS, Bluetooth, Email,	https://android.googlesource.com (accessed on 15 June 2021)
Sharejar and Source Forge	Calendar, Gallery2, and Telephony	https://sourceforge.net/p/bitweaver/bugs/ http://securityfocus.com https://sourceforge.net/p/Webcalendar/bugs
GitHub Repository	Bootstrap, Avaya Communicator, The K-9 Mail client, Space Blaster game, K-9 issue report.	https://github.com/bpellin/keepassdroid https://github.com/adrian/upm-android http://android.scap.org.cn/ http://cve.mitre.org/ https://github.com/breezedong/DNN-based SFP https://github.com/JesusFreke/smali/wiki https://github.com/tapjdey/release_qual_model https://github.com/geometer/FBReaderJ
Mobile Applications Web Applications	Connectbot, Boardgame Geek, AnkiDroid, Android Wallpaper, Quiksearchbox, Afwall, Alfresco, AndroidSync F-droid Fusion Android PHP web apps Drupal project Tech-terms	https://techterms.com/definition/repository http://pallergabor.uw.hu/androidblog/dalvik_opcodes.html https://source.android.com/devices/tech/dalvik/dalvikbytecode http://searchoracle.techtarget.com/definition/repository http://code.google.com/p/sipdroid/ https://forum.xda-developers.com/showthread.php?t=1800090 https://tinyurl.com/m722ouohttps://tinyurl.com/ya533yya https://tinyurl.com/y9vcudjthttps:/Iplay.google.comlstore/search?q=weather%20forecast http://f-droid.org! http://23.92.18.210:8080/FusionWeb/viewer.jsp http://sharlwinkhin.com/phpminer.html http://www.drupal.org/ https://techterms.com/definition/repository

4.3. RQ-3: Machine Learning Types

Supervised learning algorithms were preferred in 43 papers. The other machine learning types (i.e., unsupervised and semi-supervised) were used in four publications (two papers per each type). Figure 7 shows the distribution of ML types used in selected publications. This indicates that most of the researchers preferred supervised learning approaches when developing models for mobile applications. However, the literature also includes unsupervised fault prediction models [34] and semi-supervised fault learning models [35]. Additionally, noisy instances can be removed from the datasets to improve the overall performance of the supervised models [36].

4.4. RQ-4: Machine Learning Algorithms

In studies that do not employ deep learning techniques, for the most part, a static feature selection that is manually chosen by knowledgeable domain experts is preferred. However, we also observed that the Correlation-based Feature Selection (CFS) method was used in several studies [37–41] as the feature subset selection technique. Secondly, gain ratio attribute evaluation is used [42–44] to reduce the high-dimensionality and further improve efficiency. Alternatively, machine learning models such as Logistic Regression (LR) and Random Forest (RF) were built [45,46] for the same purpose. The different methods were noted as applying evolutionary techniques [47], statistical feature selection [48], Principal Component Analysis [49], and T-test analysis-based feature selection [50].

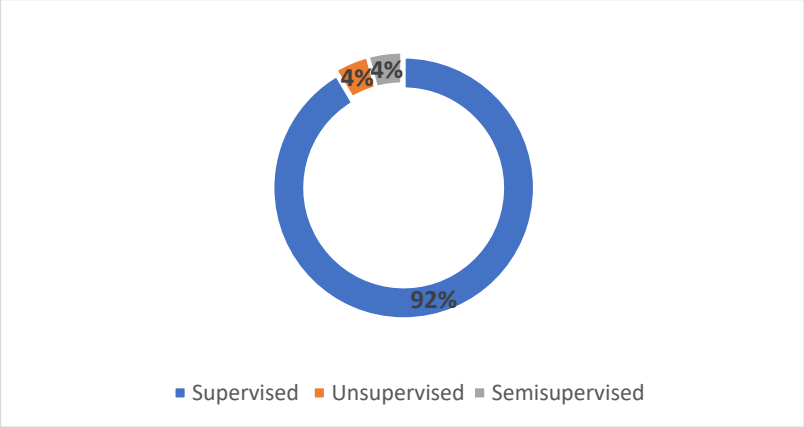

Figure 7. Distribution of Machine Learning Types.

In selected studies, eighteen machine learning methods were used. The algorithms are as follows: Naïve Bayes (NB) in 26 studies, Support Vector Machines (SVM) in 22 studies, Logistic Regression (LR) were in 19 studies, Neural Network (NN) in 18 studies, Decision Tree (DT) in 18 studies, Random Forest (RF) in 15 studies, K-nearest Neighbors (KNN) in six studies, Alpha-beta pruning (AB), K-means, and Bayesian Network (BN) in five studies, respectively, Bootstrap aggregating (Bag) and Multilayer Perceptron (MLP) in three studies, respectively, and Voting Future Intervals (VFI), DTNB, Non-Nested Generalization (NNge) and Logistic model tree (LMT) were used in two studies, respectively. Artificial Neural Network (ANN), and adaptive genetic algorithm (AGA) were used only in one study each. Figure 8 shows the distribution of algorithms. Based on this analysis, we can state that the top three applied algorithms are Naïve Bayes, Support Vector Machines, and Logistic Regression algorithms.

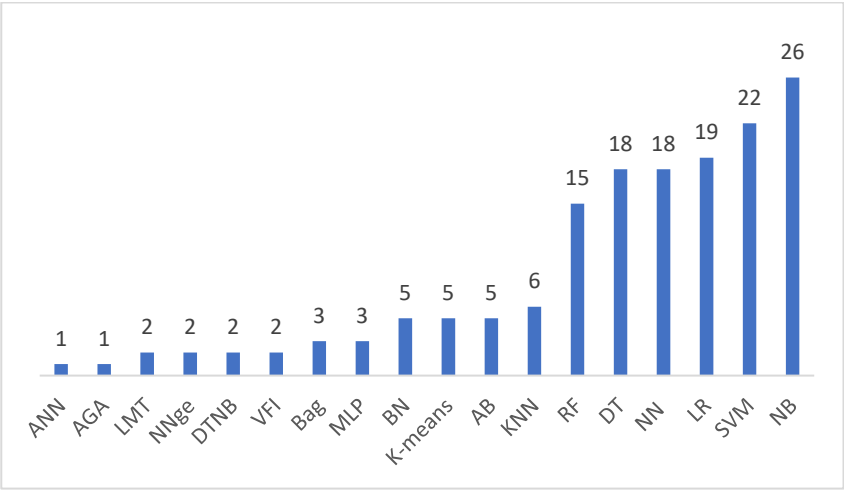

Figure 8. Machine Learning Algorithms.

4.5. RQ-5: Evaluation Metrics

We identified 19 evaluation metrics in the selected articles. The Precision, Recall, and Accuracy metrics set was used in 31 articles. Seventeen articles used ROC Curve and Area under ROC curve (AUC) metrics. F-measure was used in 10 publications. F1 score was used in four publications. Mean Absolute Error was used in four publications. The distribution of evaluation metrics is presented in Figure 9. This figure indicates that most researchers preferred the precision and recall parameters while evaluating their models. Additionally, AUC is widely used by researchers in this field.

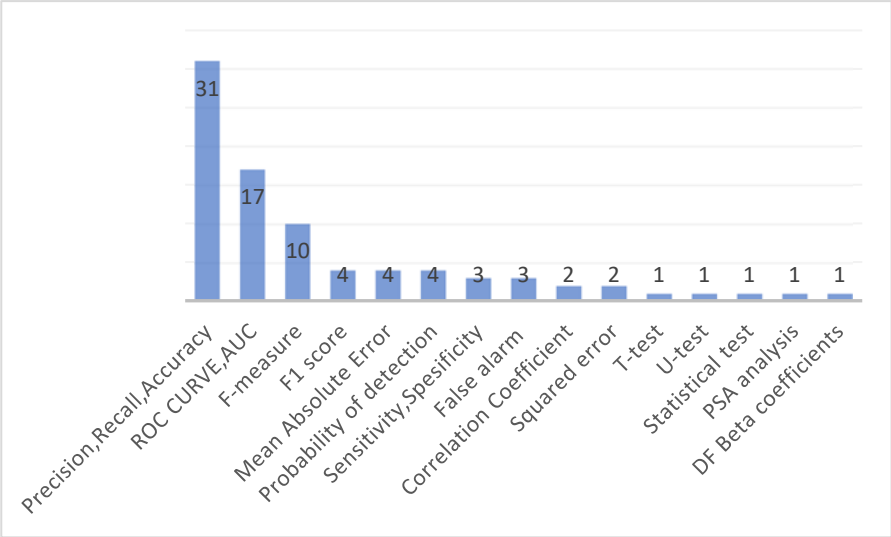

Figure 9. Evaluation metrics.

4.6. RQ-6: Validation Approaches

Eighty-eight percent of studies used k-fold cross-validation in these papers. Leave-one-out validation was applied in 12% of studies. Figure 10 represents the distribution of applied validation approaches. This figure indicates that most researchers prefer the use of K-fold cross-validation in mobile defect prediction.

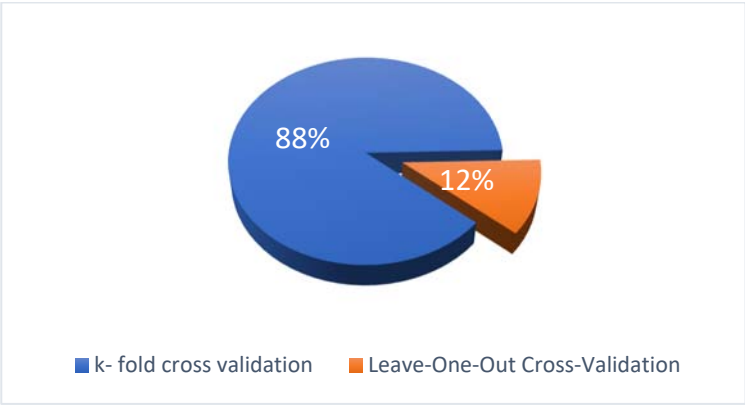

Figure 10. Distribution of validation approaches.

4.7. RQ-7: Software Metrics

We extracted all the metrics used in primary studies. As seen in selected publications, many metric types have been used. Therefore, we decided to categorize metrics. Object-oriented metrics were used in 24 publications. Procedural metrics were used in 11 publications. Web metrics were used in two publications. Process metrics were applied in two publications. Performance metrics were used in two publications. Figure 11 shows the distribution of metric types applied in selected papers. This figure indicates that most researchers preferred object-oriented metrics in mobile defect prediction studies.

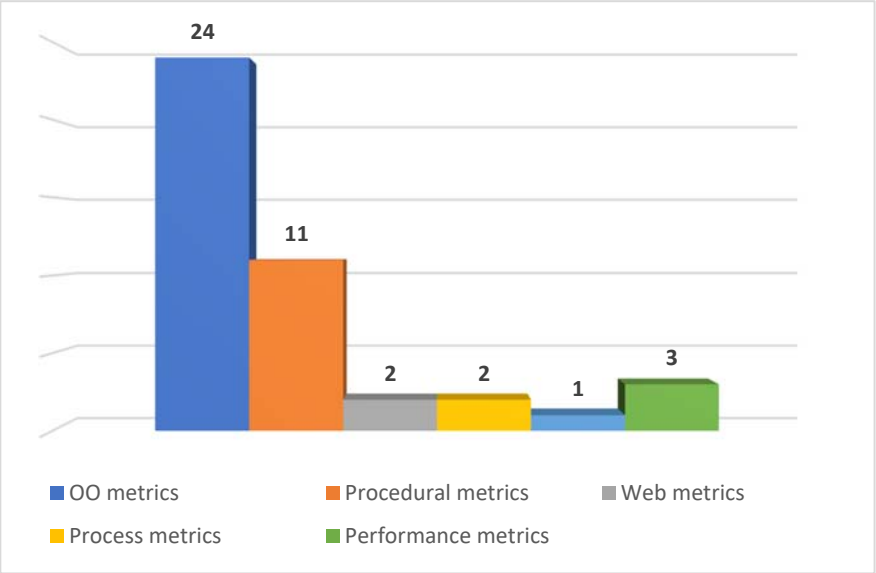

Figure 11. Distribution of metrics types.

4.8. RQ-8: The Best Algorithm

We categorized algorithms into two categories: traditional machine learning algorithms and Deep Learning algorithms. Figure 12 shows the best performing machine learning algorithms. Support Vector Machines (SVM) was identified four times as the best algorithm in publications. Random Forest (RF) and Multilayer Perceptron (MLP) were reported the best algorithm in three publications. Bayesian Network (BN), Naïve Bayes (NB), and Logistic Regression (LR) were chosen twice as the best algorithm. The other algorithms, Multiple Linear Regression (MLR), Multinominal Naïve Bayes (MNB), Co-trained Random Forest (Co Forest), Adaptive Boosting (AdaBoost), Gradient Boosting (GBDT), J48, Bootstrap aggregating (Bag), K-means++ (K-means clustering), KNN (K-nearest neighbors), Artificial Neural Network (ANN), An adaptive genetic algorithm (AGA), were reported only once as the best algorithm. We observed that ensemble techniques, namely Random Forest, Bootstrap Aggregating, Adaptive Boosting, and Gradient Boosting are also used.

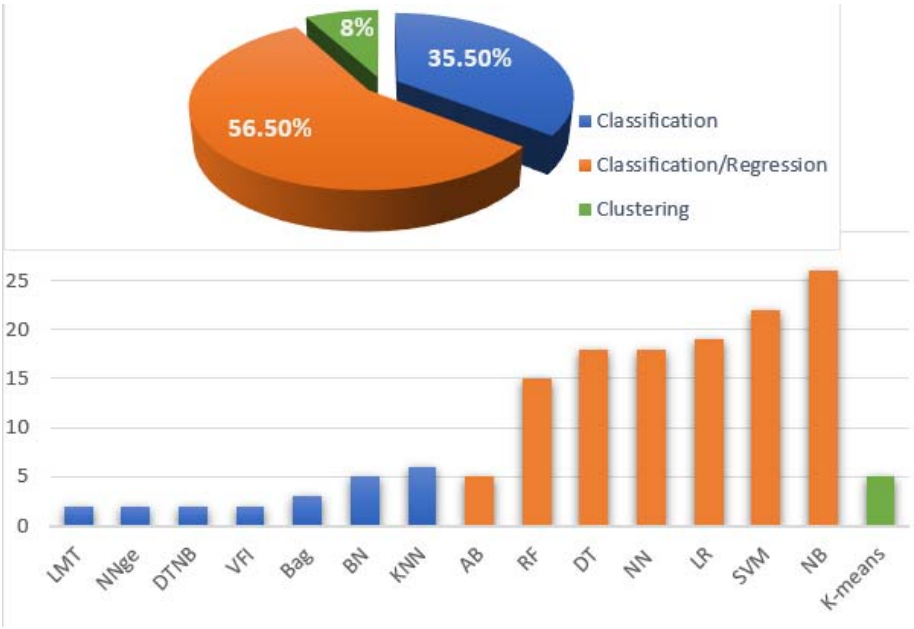

Figure 12. Machine Learning Algorithms Performance.

Figure 13 shows the distribution of Deep Learning algorithms. Long Short-Term Memory (LSTM) was specified in two publications as the best algorithm. Based on this analysis, we can state that SVM, MLP, and RF are the top three shallow learning algorithms in terms of performance and LSTM is the most important deep learning algorithm among other deep learning algorithms.

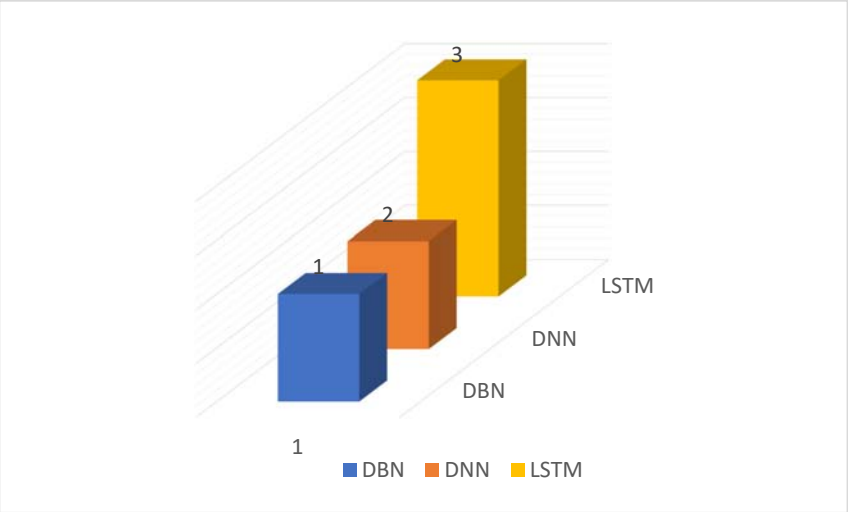

Figure 13. Distribution of Deep Learning Algorithms.

4.9. RQ-9: Challenges

In this section, we present the main challenges and proposed solutions reported in mobile defect prediction studies. Table 6 shows the main challenges and possible solutions with the references.

Table 6. Challenges and possible solutions.

Challenges	Proposed Solutions	Reference
Metric selection limitations for mobile software	Use alternate code and process metrics	[3,18]
Faults in Android data	Remove faults	[9]
Limited mobile app repository	Use of public repository	[11,27,28,36]
Repeated data/code in the project	Domain Adaptation	[26]
Small dataset problem	Not mentioned	[22]
Different programming language problem	Defect prediction only GIT open-source Android, Java, and C++ uncertain	[8,10,26]
Modeling problem	Not mentioned	[4,11,30]
Different platforms and languages	Not mentioned	[18,21]
Extensive datasets	Not mentioned	[16]
Not fully automated	Manually code, log, bug, and review control	[11]
Imbalance Class problem	Sampling methods, Under sampling methods	[7,12,22]
Manual feature engineering	Not mentioned	[26]

5. Discussion

In this section, we present the general discussion and validity considerations of this systematic literature review.

5.1. General Discussion

This study aims to collect, synthesize, and evaluate mobile application defect prediction publications using machine learning techniques. To the best of our knowledge, there has been no similar SLR paper published on this topic yet. Therefore, we performed this SLR study and aimed to answer some research questions that we defined at the beginning of this research. We believe that the observations and suggestions will pave the way for further research and help both practitioners and researchers in this field. Responses to research questions are briefly discussed as follows:

RQ1—We noticed that most papers addressed the Android platform but the Windows mobile operating system was discussed in only 2% of the studies. We also did not see any paper that focused on the iOS platform. The reason is probably related to the open source code bases of Android-based applications, which supported the researchers in a way that they were able to calculate the software metrics and collect the defect information from different publicly available repositories. Since many mobile platforms use the Android operating system, many researchers prefer to perform experiments on this platform.

RQ2—Many datasets have been stored in Github or git repositories for defect prediction. These repositories are widely used by practitioners and researchers, therefore, the available datasets are mostly located in these platforms. This is also related to the open source nature of Android applications; they are mostly hosted in these platforms. There were also a few datasets that used other platforms, however, their percentage was lower compared to the use of Github-related repositories.

RQ3—Most of the studies used supervised learning approaches; they were limited number of papers that used unsupervised and semi-supervised learning techniques. The reason is that most researchers were probably able to obtain the available defect information from the publicly available repositories and therefore, they aimed to build supervised learning models instead of unsupervised or semi-supervised learning models. However, it is also possible to carry out some experiments in the context of available defect information by simulating different scenarios. There is still some room for further research on the use of these less preferred machine learning types.

RQ4—Naïve Bayes (NB), Support Vector Machines (SVM), and Logistic Regression algorithms are the most preferred algorithms. The reason is most probably that researchers preferred the widely used machine learning algorithms such as SVM and NB in their experiments. Previously, it has been also demonstrated that NB provides high performance in software defect prediction [51], and therefore, it might have been preferred in the mobile application defect prediction studies as well.

RQ5—Most of the papers used Precision, Recall, and Accuracy metrics to evaluate the performance of the models and also the Area Under ROC curve (AUC) metric was preferred by researchers. These metrics are widely used in machine learning studies and therefore, researchers probably selected these metrics. Accuracy is not a good metric for defect prediction studies because these datasets are imbalanced and the accuracy metric cannot be used alone to judge the performance of the models; it must be used together with other metrics such as precision and recall.

RQ6—Most studies used the k-fold cross-validation strategy for the evaluation of the model performance. This is also the widely used evaluation approach among machine learning approaches and therefore, researchers might have preferred to use this strategy. There are also other alternatives such as leave-one out technique; however, k-fold cross-validation was applied by most researchers. There is also possibility to perform k-fold n times, which can be called k*n cross-validation; however, the use of this strategy in these papers was quite limited, although this approach can present more statistically sound results.

RQ7—Most of the studies used object-oriented metrics. This is probably due to the widespread adoption of object-oriented programming paradigms in the software industry. However, new metrics can be proposed and evaluated by researchers for mobile applications. This might be a potential research topic for researchers.

RQ8—Support Vector Machines (SVM), Random Forest (RF), Multilayer Perceptron (MLP) were among the best performing algorithms. Among deep learning algorithms, LSTM provided the best performance. Since the training of deep learning models requires more time and data, in some cases, researchers and practitioners can consider the scale of the dataset before building the prediction model. If traditional machine learning algorithms (i.e., shallow learning) can provide high performance, more complex algorithms such as deep learning might not be needed.

RQ9—Several challenges were mentioned to answer this research question. We extracted these challenges from the papers if they were mentioned. However, there is a possibility that authors might not have discussed the challenges in the paper explicitly. In such cases, we were unable to add those challenges. If the challenge has not been experienced by researchers and mentioned as future work, they were also not included. There might be additional challenges that are missing in this paper; however, we aimed to collect them from the available literature.

5.2. Threats to Validity

We selected publications from six digital platforms using our search criteria and also conducted a snowballing process. Authors held several meetings to minimize the researcher bias. However, there might be some papers in some electronic databases that we have missed in this research. Additionally, new papers are also published very frequently and therefore, we might have missed some new papers published recently. Another threat is the use of the search criteria. There might be more synonyms that could have been used in this research and we have missed some papers due to this issue.

6. Conclusions and Future Work

This study presented the results of a systematic literature review on mobile fault prediction using machine learning. A total of 721 publications were retrieved from electronic databases, and after study selection criteria, 47 publications were selected. The selected publication is classified based on platforms, datasets, machine learning types, machine learning algorithms, evaluation metrics, validation approaches, software metrics, best machine learning and deep learning algorithms, challenges and gaps, and the corresponding results are reported. The Android platform was mostly preferred by researchers. Furthermore, there exists a limited number of repositories and datasets for mobile defect prediction studies. Most researchers used object-oriented metrics in mobile defect prediction. Most of the studies used supervised learning algorithms instead of unsupervised and semi-supervised learning algorithms. This means that there is still a potential for further research using unsupervised and semi-supervised learning for mobile defect prediction. We are planning to build novel prediction models using these algorithms for the Android platform.

Author Contributions: Conceptualization: M.J., A.A. and C.C.; data curation: M.J.; formal analysis: M.J., A.A., C.C. and A.M.; investigation: M.J., A.A., C.C. and A.M.; methodology: M.J., A.A., C.C. and A.M.; project administration: A.A. and C.C.; resources: M.J., A.A., C.C. and A.M.; supervision: A.A. and C.C.; validation: M.J., A.A., C.C. and A.M.; writing—original draft: M.J., A.A., C.C. and A.M.; writing—review and editing: M.J., A.A., C.C. and A.M. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by Molde University College-Specialized Univ. in Logistics, Norway for the support of Open Access fund.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Kaur, A.; Kaur, K. An investigation of the accuracy of code and process metrics for defect prediction of mobile applications. In Proceedings of the 2015 4th International Conference on Reliability, Infocom Technologies and Optimization (ICRITO) (Trends and Future Directions), Noida, India, 2–4 September 2015.
2. Kha1id, H.E.; Shihab, M. What do mobile app users complain about? a study on free ios apps. *IEEE Softw.* **2015**, *32*, 222.
3. Harman, M.; Jia, Y.; Zhang, Y. App store mining and analysis: MSR for app stores. In Proceedings of the 2012 9th IEEE Working Conference on Mining Software Repositories (MSR), Zurich, Switzerland, 2–3 June 2012; pp. 108–111.
4. Xia, X.; Shihab, E.; Kamei, Y.; Lo, D.; Wang, X. Predicting crashing releases of mobile applications. In Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, Ciudad Real, Spain, 8–9 September 2016; pp. 1–10.
5. Avizienis, A.; Laprie, J.-C.; Randell, B.; Landwehr, C. Basic concepts and taxonomy of dependable and secure computing. *IEEE Trans. Dependable Secur. Comput.* **2004**, *1*, 11–33. [CrossRef]
6. Malhotra, R. An empirical framework for defect prediction using machine learning techniques with Android software. *Appl. Soft Comput.* **2016**, *49*, 1034–1050. [CrossRef]
7. Hall, T.; Beecham, S.; Bowes, D.; Gray, D.; Counsell, S. systematic literature review on fault prediction performance in software engineering. *IEEE Trans. Softw. Eng.* **2011**, *38*, 1276–1304. [CrossRef]
8. Mishra, A.; Shatnawi, R.; Catal, C.; Akbulut, A. Techniques for Calculating Software Product Metrics Threshold Values: A Systematic Mapping Study. *Appl. Sci.* **2021**, *11*, 11377. [CrossRef]
9. Catal, C.; Diri, B. A systematic review of software fault prediction studies. *Expert Syst. Appl.* **2009**, *36*, 7346–7354. [CrossRef]
10. Malhotra, R.; Jain, A. Software fault prediction for object-oriented systems: A systematic literature review. *ACMSIGSOFT Softw. Eng.* **2011**, *36*, 1–6. [CrossRef]
11. Malhotra, R. A systematic review of machine learning techniques for software fault prediction. *Appl. Soft Comput.* **2015**, *27*, 504–518. [CrossRef]
12. Radjenovic, D.; Heriko, M. Software fault prediction metrics: A systematic literature review. *Inf. Softw. Technol.* **2013**, *55*, 1397–1418. [CrossRef]
13. Misirli, A.T.; Bener, A.B. A mapping study on Bayesian networks for software quality prediction. In Proceedings of the 3rd International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering, Hyderabad, India, 3 June 2014; pp. 7–11.
14. Murillo-Morera, J.; Quesada-López, C.; Jenkins, M. *Software Fault Prediction: A Systematic Mapping Study*; CIbSE: London, UK, 2015; p. 446.
15. Özakıncı, R.; Tarhan, A. Early software defect prediction: A systematic map and review. *J. Syst. Softw.* **2018**, *144*, 216–239. [CrossRef]
16. Son, L.H.; Pritam, N.; Khari, M.; Kumar, R.; Phuong, P.T.M.; Thong, P.H. Empirical Study of Software Defect Prediction: A Systematic Mapping. *Symmetry* **2019**, *11*, 212. [CrossRef]
17. Najm, A.; Zakrani, A.; Marzak, A. Decision Trees Based Software Development Effort Estimation: A Systematic Mapping Study. In Proceedings of the 2019 International Conference of Computer Science and Renewable Energies (ICCSRE), Agadir, Morocco, 22–24 July 2019.
18. Alsolai, H.; Roper, M. A systematic literature review of machine learning techniques for software maintainability prediction. *Inf. Softw. Technol.* **2020**, *119*, 106214. [CrossRef]
19. Auch, M.; Weber, M.; Mandl, P.; Wolff, C. Similarity-based analyses on software applications: A systematic literature review. *J. Syst. Softw.* **2020**, *168*, 110669. [CrossRef]
20. Degu, A. Android application memory and energy performance: Systematic literature review. *IOSR J. Comput. Eng.* **2019**, *21*, 20–32.
21. Kaur, A.; Kaur, K. Systematic literature review of mobile application development and testing effort estimation. *J. King Saud Univ.Comput. Inf. Sci.* **2018**, *34*, 1–15. [CrossRef]
22. Del Carpio, A.F.; Angarita, L.B. Trends in Software Engineering Processes using Deep Learning: A Systematic Literature Review. In Proceedings of the 2020 46th Euromicro Conference on Software Engineering and Advanced Applications (SEAA), Portorož, Slovenia, 26–28 August 2020; pp. 445–454.
23. Kaur, A. A Systematic Literature Review on Empirical Analysis of the Relationship between Code Smells and Software Quality Attributes. *Arch. Comput. Methods Eng.* **2019**, *27*, 61267–61296. [CrossRef]
24. Kaya, A.; Keceli, A.S.; Catal, C.; Tekinerdogan, B. Model analytics for defect prediction based on design-level metrics and sampling techniques. In *Model Management and Analytics for Large Scale Systems*; Academic Press: Cambridge, MA. USA, 2020; pp. 125–139.
25. Kaur, A.; Kaur, K.; Kaur, H. Application of machine learning on process metrics for defect prediction in mobile application. In *Information Systems Design and Intelligent Applications*; Springer: New Delhi, India, 2016; pp. 81–98.
26. Zhao, K.; Xu, Z.; Yan, M.; Tang, Y.; Fan, M.; Catolino, G. Just-in-time defect prediction for Android apps via imbalanced deep learning model. In Proceedings of the 36th Annual ACM Symposium on Applied Computing, Online, 22–26 March 2021; pp. 1447–1454.

27. Sewak, M.; Sahay, S.K.; Rathore, H. Assessment of the Relative Importance of different hyper-parameters of LSTM for an IDS. In Proceedings of the 2020 IEEE REGION 10 CONFERENCE (TENCON), Osaka, Japan, 16-19 November 2020; pp. 414–419.
28. Pandey, S.K.; Mishra, R.B.; Tripathi, A.K. Machine learning based methods for software fault prediction: A survey. *Expert Syst. Appl.* **2021**, *172*, 114595. [CrossRef]
29. Bhavana, K.; Nekkanti, V.; Jayapandian, N. Internet of things enabled device fault prediction system using machine learning. In *International Conference on Inventive Computation Technologies*; Springer: Cham, Switzerland, 2019; pp. 920–927.
30. Pandey, S.K.; Tripathi, A.K. DNNAttention: A deep neural network and attention based architecture for cross project defect number prediction. *Knowl. Based Syst.* **2021**, *233*, 107541. [CrossRef]
31. Kitchenham, B.; Brereton, O.P.; Budgen, D.; Turner, M.; Bailey, J.; Linkman, S. Systematic literature reviews in software engineering—A systematic literature review. *Inf. Softw. Technol.* **2009**, *51*, 7–15. [CrossRef]
32. Kang, Z.; Catal, C.; Tekinerdogan, B. Machine learning applications in production lines: A systematic literature review. *Comput. Ind. Eng.* **2020**, *149*, 106773. [CrossRef]
33. Kitchenham, B.; Charters, S. *Guidelines for Performing Systematic Literature Reviews in Software Engineering*; Technical Report; EBSE: Goyang-si, Korea, 2007.
34. Catal, C.; Sevim, U.; Diri, B. Metrics-driven software quality prediction without prior fault data. In *Electronic Engineering and Computing Technology*; Springer: Dordrecht; The Netherlands, 2010; pp. 189–199.
35. Catal, C. A Comparison of Semi-Supervised Classification Approaches for Software Defect Prediction. *J. Intell. Syst.* **2014**, *23*, 75–82. [CrossRef]
36. Alan, O.; Catal, C. Thresholds based outlier detection approach for mining class outliers: An empirical case study on software measurement datasets. *Expert Syst. Appl.* **2011**, *38*, 3440–3445. [CrossRef]
37. Ricky, M.Y.; Purnomo, F.; Yulianto, B. Mobile application software defect prediction. In Proceedings of the 2016 IEEE Symposium on Service-Oriented System Engineering (SOSE), Oxford, UK, 29 March–2 April 2016; pp. 307–313.
38. Biçer, M.S.; Diri, B. Predicting defect-prone modules in web applications. In *International Conference on Information and Software Technologies*; Springer: Cham, Switzerland, 2015; pp. 577–591.
39. Malhotra, R.; Sharma, A. Empirical assessment of feature selection techniques in defect prediction models using web applications. *J. Intell. Fuzzy Syst.* **2019**, *36*, 6567–6578. [CrossRef]
40. Ramakrishnan, R.; Kaur, A. An empirical comparison of predictive models for web page performance. *Inf. Softw. Technol.* **2020**, *123*, 106307. [CrossRef]
41. Kang, D.; Bae, D.H. Software fault prediction models for web applications. In Proceedings of the 2010 IEEE 34th Annual Computer Software and Applications Conference Workshops, Seoul, Korea, 19–23 July 2010; pp. 51–56.
42. Catolino, G.; Di Nucci, D.; Ferrucci, F. Cross-project just-in-time bug prediction for mobile apps: An empirical assessment. In Proceedings of the 2019 IEEE/ACM 6th International Conference on Mobile Software Engineering and Systems (MOBILESoft), Montreal, QC, Canada, 25–26 May 2019; pp. 99–110.
43. Kumar, A.; Chugh, R.; Girdhar, R.; Aggarwal, S. Classification of faults in Web applications using machine learning. In Proceedings of the 2017 International Conference on Intelligent Systems, Metaheuristics & Swarm Intelligence, Hong Kong, China, 25–27 March 2017; pp. 62–67.
44. Bhandari, G.P.; Gupta, R. Fault Prediction in SOA-Based Systems Using Deep Learning Techniques. *Int. J. Web Serv. Res.* **2020**, *17*, 1–19. [CrossRef]
45. Cui, J.; Wang, L.; Zhao, X.; Zhang, H. Towards predictive analysis of android vulnerability using statistical codes and machine learning for IoT applications. *Comput. Commun.* **2020**, *155*, 125–131. [CrossRef]
46. Shar, L.K.; Briand, L.C.; Tan, H.B.K. Web Application Vulnerability Prediction Using Hybrid Program Analysis and Machine Learning. *IEEE Trans. Dependable Secur. Comput.* **2014**, *12*, 688–707. [CrossRef]
47. Malhotra, R.; Khurana, A. Analysis of evolutionary algorithms to improve software defect prediction. In Proceedings of the 2017 6th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions) (ICRITO), Noida, India, 20–22 September 2017; pp. 301–305.
48. Pang, Y.; Xue, X.; Wang, H. Predicting vulnerable software components through the deep neural network. In Proceedings of the 2017 International Conference on Deep Learning Technologies, Chengdu, China, 2–4 June 2017; pp. 6–10.
49. Dehkordi, M.R.; Seifzadeh, H.; Beydoun, G.; Nadimi-Shahraki, M.H. Success prediction of android applications in a novel repository using neural networks. *Complex Intell. Syst.* **2020**, *6*, 573–590. [CrossRef]
50. Padhy, N.; Satapathy, S.C.; Mohanty, J.; Panigrahi, R. Software reusability metrics prediction by using evolutionary algorithms: The interactive mobile learning application RozGaar. *Int. J. Knowl.-Based Intell. Eng. Syst.* **2018**, *22*, 261–276. [CrossRef]
51. Menzies, T.; Greenwald, J.; Frank, A. Data Mining Static Code Attributes to Learn Defect Predictors. *IEEE Trans. Softw. Eng.* **2006**, *33*, 2–13. [CrossRef]