A Literature Review of Using Machine Learning in Software Development Life Cycle Stages

SAAD SHAFIQ, (Graduate Student Member, IEEE), **ATIF MASHKOOR,**
CHRISTOPH MAYR-DORN, AND ALEXANDER EGYED, (Senior Member, IEEE)
Institute of Software Systems Engineering, Johannes Kepler University, 4040 Linz, Austria

Corresponding author: Saad Shafiq (saad.shafiq@jku.at)

ABSTRACT The software engineering community is rapidly adopting machine learning for transitioning modern-day software towards highly intelligent and self-learning systems. However, the software engineering community is still discovering new ways how machine learning can offer help for various software development life cycle stages. In this article, we present a study on the use of machine learning across various software development life cycle stages. The overall aim of this article is to investigate the relationship between software development life cycle stages, and machine learning tools, techniques, and types. We attempt a holistic investigation in part to answer the question of whether machine learning favors certain stages and/or certain techniques.

INDEX TERMS Software engineering, machine learning, literature review.

I. INTRODUCTION

The software engineering (SE) community is continuously looking for better and more efficient ways of building high-quality software systems. However, in practice, the strong emphasis on time to market tends to ignore many, well-known SE recommendations. That is, practitioners focus more on programming as compared to requirements gathering, planning, specification, architecture, design, and documentation – all of which are ultimately known to greatly benefit the cost-effectiveness and quality of software systems. Lack of human resources is often cited as the main reason for doing so. Herein lies the great potential for machine learning (ML) since its algorithms are proven to be most befitting to problem domains that aim to replicate human behavior. Hence, it stands to reason that human-centric SE activities should also benefit from ML [1].

The growing demand for agility and the ability to solve complex problems in SE has already led researchers to explore the potential of ML in this field. To date, ML has

The associate editor coordinating the review of this manuscript and approving it for publication was Mahmoud Elish.

many demonstrated benefits. Applications of ML for SE range from resolving ambiguous requirements to predicting software defects [2]. For example, Sultanov *et al.* [3] used reinforcement learning (a type of ML) on understanding the relationships among software requirements at different levels of abstraction. Their approach shows how ML can automatically generate traceable links between high-level and low-level requirements. However, ML is not a single technique but rather an assortment of techniques. The challenge of using ML for SE is thus not only about finding the right way of solving the problem but also about comparing various ML techniques and their potential. For example, several researchers have explored predictions in order to better estimate the time to market for software products. For this purpose, various ML techniques were used and compared, e.g., artificial neural networks, rule induction, case-based reasoning, support vector machines, regression-based decision trees, and random forest [4]–[7].

In many areas of science and engineering, such as image recognition or autonomous driving, ML has already revolutionized development. The applications of ML to SE are also increasing significantly, which is evident through the

exponential growth in the number of articles on ML for SE being published every year. Consequently, it is of interest to understand, which software development life cycle (SDLC) stages benefit the most from this trend; or even to understand which ML techniques are most suitable for which SDLC stage(s). This leads to the motivation of conducting this study.

In this article, we provide a *bird's-eye view* on the current state-of-the-art regarding the causal relationship between ML and SDLC stages and suggest the open areas of research where more primary studies are needed. The fairly broad scope of this study is *by design*. While this article sets out to explore the causal relationship between machine learning and SDLC stages in the form of a *literature review*, it should be noted that some specialized studies already exist, e.g., ML for automated software testing [8]. Similar to exploratory studies conducted in the past, such as Bindewald *et al.* [9], our review is based on the quantitative analysis of the articles present in the literature addressing the application of ML to various SDLC stage(s).

The rest of the article is organized as follows. The related work is discussed in Section II. Section III presents a brief introduction to ML. Section IV explains the research methodology and protocol followed in the study. Results of the study are discussed in Section V. Further analysis on the presented state-of-the-art is discussed in Section VI. Section VII presents challenges, limitations, and future research directions of this field. Section VIII discusses different threats to the validity of the presented results. The article is concluded in Section IX.

II. RELATED WORK

Some studies, e.g., [8], [10]–[12], have already analyzed the application of ML in SE in the past. Durelli *et al.* [8] conducts a systematic mapping study on the application of ML for software testing. The study highlights the use of ML techniques in various software testing activities such as test-case generation and oracle construction. Results of the study show that a vast majority of articles employ supervised learning, such as ANN and DT, to solve testing-related problems. Moreover, the key advantages and disadvantages of using ML for software testing are discussed. Mainly, the advantage of ML techniques is their scalability and efficient application to large-scale and complex software systems. The disadvantage, on the other hand, is the unavailability of data that fits well with the learning process.

Fajardo *et al.* [10] provides an extensive overview of applying data mining techniques to SE tasks including open issues and recommendations for improvements. The study provides a comprehensive list of data mining techniques applicable in SE, e.g., aspects related to clustering, regression, and performance metrics. Moreover, the study highlights some widely used datasets of SE employed in the data mining articles and states key advantages of mining SE data.

Wan *et al.* [11] performed a survey by interviewing 14 people from 3 companies and 342 respondents from 26 countries. The aim of the study was to understand and highlight the key differences in the software development practices followed in building ML and non-ML software systems. Results suggested that ML engineers should focus on handling the uncertain randomness of ML systems and work on employing version control systems specifically for ML applications in order to improve reproducibility.

Zhang *et al.* [12] conducted the research focusing on the application of ML in SE. The study provides a comprehensive list of SE tasks whose problems can be addressed using ML techniques. The study also emphasizes the fact of SE to be a highly fertile area to explore with regards to applying ML techniques by formulating SE tasks as learning problems and addressed using ML techniques.

In contrast to the aforementioned focused studies, our study provides a broader context and a comprehensive list of articles that help identify the relationship between various ML techniques and SDLC stages. We also provide the relationships of ML types, tools, and techniques with respect to SE stages, which help better understanding the current progress of the adoption of ML for SE.

III. INTRODUCTION TO ML

ML has evolved drastically over the recent years and is now being employed on a global scale. As a subset of artificial intelligence, ML has been considered vital when developing software systems for domains such as speech/image recognition [13] or automotive [14]. ML techniques have also been used to address various SE issues and activities. Most commonly, ML has been employed in defect prediction, effort estimation, and identifying patterns and similarities in the source code.

The ML techniques are essentially targeted to solve problems, which can often become hard to analyze by people causing misinformation [15]. These problems have various types, which can be categorized as ML types. ML types generally include supervised, unsupervised, and reinforcement learning. Most of the applications of ML consist of problems that can be deemed of type supervised learning. It refers to learning from features along with their known class labels. Then, predicting the class labels from new unseen features. These problems are also often categorized as classification problems.

Arguably, ML techniques can also be classified into two divisions. 1) Traditional ML techniques that include decision trees (DT), random forest (RF), linear regression, logistic regression (LR), support vector machines (SVM), k-nearest neighbors (KNN), and naive bayes (NB). 2) Neural network-based ML techniques that include artificial neural network (ANN), recurrent neural network (RNN), and convolutional neural network (CNN). Deep learning (DL) – also known as deep neural networks (DNN) – is a subset of ANN. DL was introduced mainly to address the data scalability problems such as handling high-dimensional and large-scale datasets. Structurally, instead of comprising a single hidden layer within the input and output layer as in ANN, RNN and CNN techniques are composed of multiple hidden layers

of interconnected neurons. The processing inside the hidden layers is based on the principle of weighted connections. In general, each hidden neuron is comprised of predetermined weight and bias values, which are later adjusted during the training process until the desired output is reached. Lastly, the output layer holding the acquired weight and bias values represents the solution to the given problem.

IV. RESEARCH METHODOLOGY

In order to direct the study, we followed the Goal, Question, and Metric (GQM) paradigm suggested by Basili *et al.* [16]. The aim of the GQM paradigm is to guide the study by specifying its goals in order to have an objective-oriented data extraction process. It also helps in tracing goals to formulated questions leading to better interpretation of the data in line with goals stated before.

A. GOALS

The overall aim of this study is to evaluate the relation between ML and SDLC stages. The considerably broader aim of this study differentiates it from other similar studies, such as the one by Durelli *et al.* [8], which have quite a narrow focus. Following are the goals formulated for this study.

G1. To identify the susceptibility of various ML types and techniques to SDLC stages

G2. To understand the maturity of research in this area

G3. To identify the demographics in this area

G4. To understand the implications, challenges, limitations, and future research directions in this area

   The first three goals lead to the research questions discussed in the following subsection. Due to the descriptive and elaborate nature of the fourth goal, we decided to thoroughly discuss it in Sections VI and VII.

B. QUESTIONS

In order to meet the outlined goals, we have formulated concrete questions and identified suitable metrics (quantifiable attributes). The questions and metrics (emphasized) are explained as follows:

G1. The susceptibility of various ML types and techniques to SDLC stages

Q1.1 What *SDLC stages* are being focused on by academic and industrial researchers in this area?

   Rationale: Our interest is to understand what SDLC stage the researchers tend to focus on, whether, a particular SDLC stage or the amalgamation of two or more. The SDLC stages are based on, but not limited to, the knowledge areas mentioned in SWEBOK [17] characterizing the practice of SE, e.g., Software Requirements, Software Design, or Software Maintenance.

Q1.2 What are the *applications of ML* for SE?

   Rationale: We are interested to know about the specific applications of ML that exist in SE, e.g., whether an ML technique was used to automate test case generation or to predict potential bugs in the system.

Q1.3 What *type of ML and its techniques* are being employed for SE?

   Rationale: We are interested to know whether a particular type/technique was consistently employed for a specific life cycle stage. The type of ML refers to how the models have been trained, e.g., supervised, semi-supervised, or unsupervised. Whereas the ML techniques are the algorithms used for classification or clustering problems, e.g., SVM, RF, or ANN.

G2. The maturity of research in the area

Q2.1 What is the *contribution facet* of the articles?

   Rationale: The contribution facet refers to the novel contributions made by the researchers in the articles. It partially corroborates the attributes provided by Banerjee *et al.* [18] and Petersen *et al.* [19], and are supplemented by our own views obtained by analyzing the extracted articles. The attributes are defined as follows:

   - **Tool:** Article proposing a new tool or improving an existing one and describing its evaluation.
   - **Approach/method:** Article proposing a new approach/method or improving the existing one.
   - **Model/framework:** Article introducing a new model/framework or improving the existing one.
   - **Algorithm/process:** Article proposing a new algorithm/SE process or improving the existing one.
   - **Comparative analysis:** Article evaluating different approaches and reporting results of the comparative study.

Q2.2 What is the *research facet* of the articles?

   Rationale: The research facet of an article refers to the maturity of the research in terms of empirical evidence provided in the article or whether an article was proposing a solution or evaluating an existing approach. The research facet is defined as follows:

   - **Evaluation:** Article evaluating or validating the proposed approach using empirical methods.
   - **Knowledge:** Article describing the experiences and opinions of authors on the existing approaches.
   - **Solution:** Article proposing a new solution and describing its applicability with the help of examples and arguments.

   We further explored the evaluation facet in order to understand the empirical methods employed in the articles.

Q2.3 What *datasets* are commonly employed in the articles?

   Rationale: We are interested to know about the datasets that are most commonly used to evaluate the research results in the domain of ML for SE.

G3. The demographics of research in the area

Q3.1 What are the *trends in terms of years* of publications in the area?

   Rationale: The trends in terms of years refer to the number of publications varying from a year to another.

Here, we want to assess how active this research area is.

Q3.2 What are the *highest publishing venues* of the area?
Rationale: We are interested to know about the venues, which have the highest publications with respect to the area of ML for SE.

C. ARTICLES EXTRACTION

1) QUERY FORMULATION

In order to search for relevant literature, following the guidelines proposed by Petersen *et al.* [19], we devised a query that uses a two-element PICO search. **Problem 'P':** (requirement, specification, design, model, analysis, architecture, implementation, code, test, verification, validation, maintenance), and **Intervention 'I':** (machine learning, deep learning). We have not considered Comparison 'C' and Outcome 'O' as this is out of the scope of this study. Following is the resultant query that was eventually used in all digital libraries:

("machine learning" OR "deep learning") AND software AND requirement* OR specification* OR design* OR model* OR analysis OR architecture OR implementation OR code OR test* OR verification OR validation OR maintenance[1]

2) DIGITAL LIBRARIES

The query was applied to titles and abstracts of articles in five well-known digital libraries: ACM Digital Library,[2] IEEEXplore,[3] ScienceDirect,[4] Springer,[5] and Web of Science.[6] According to [20], these digital libraries are among the most popular sources in computer science and engineering that ensure high coverage of potentially relevant studies. We did not include Google Scholar[7] in our study as the search results of Google Scholar tend to be repetitive with respect to results from the included digital libraries, and its unique contribution to the search process is unclear [20].

3) TIME PERIOD

We scope the time period of related studies published from 1991 to 2020. The earliest paper we could find in our study was published in 1991, hence the starting time. We conducted the search in Q1 2021 and made sure that the results are reproducible until 2020, hence the ending time.

4) ARTICLES SELECTION

All repositories, except Springer, returned the number of articles as shown in Fig. 1. Springer initially yielded 4,502 articles as a result of the query; however, most of these articles were quite irrelevant to the scope of our study even after applying filters, such as "Computer Science" as discipline,

and "SE" and "AI" as sub-disciplines, to reduce the search space. We then went through the titles and abstracts of the articles (if the goal of the article is unclear from the title) and stopped the search process when the first page with all irrelevant articles was reached. This resulted in 46 articles.

In total, we extracted 565 articles, as shown in Fig. 1. However, many of them were duplicated as one article may appear in many digital libraries. We then removed duplicates, which resulted in 501 unique articles. The articles then underwent a screening process and were scrutinized based on the following inclusion criteria.

1) Articles that were relevant to the scope, i.e., relevance and appropriateness of the article correspond to the research goals of the study, were included.
2) Articles that were available in the full-text format were included.
3) Articles demonstrating well-established empirical soundness were included.
4) Articles of more than a single page were included.
5) Articles that were peer-reviewed were included.
6) Articles that were entirely written in English were included.

Consequently, a total of 263 relevant articles were selected and included in the final pool. Fig. 1 shows the overall article extraction process including the number of articles extracted from each repository and the final pool.

5) TOOL

Conducting a literature review is a tedious and time-consuming task. It usually involves the search, collection, filtration, and classification of a huge amount of papers. Without a helping tool, this is a very difficult endeavor. In this work, we used Mendeley[8] and Google Sheets.[9] These tools helped us in importing, organizing, and analyzing search results.

D. CLASSIFICATION SCHEME

We later defined a classification scheme to ensure accurate assessment of attributes [19]. The generalized attributes obtained were then sorted by the authors of this study based on the knowledge areas provided in SWEBOK [17]. In fact, the knowledge areas mentioned in SWEBOK were not strictly used in the categorization but merely employed as a defining factor to providing a high-level abstraction of attributes that represented the set of articles. However, we referred to the following knowledge areas while devising the categories in this study: 1) software requirements, 2) software design, 3) software construction, 4) software quality, and 5) software maintenance. During the article sorting process, certain articles were found to be equivocal. In such cases, we associated those attributes to the articles that received majority votes from the authors of this study. To get a better understanding, a graphical representation of the workflow starting from the

---

[1]Asterisk (*) is a wildcard that refers to zero or more characters in a word
[2]https://dl.acm.org/
[3]https://ieeexplore.ieee.org/
[4]https://www.sciencedirect.com/
[5]https://www.springer.com/
[6]https://apps.webofknowledge.com/
[7]https://scholar.google.com

[8]https://www.mendeley.com
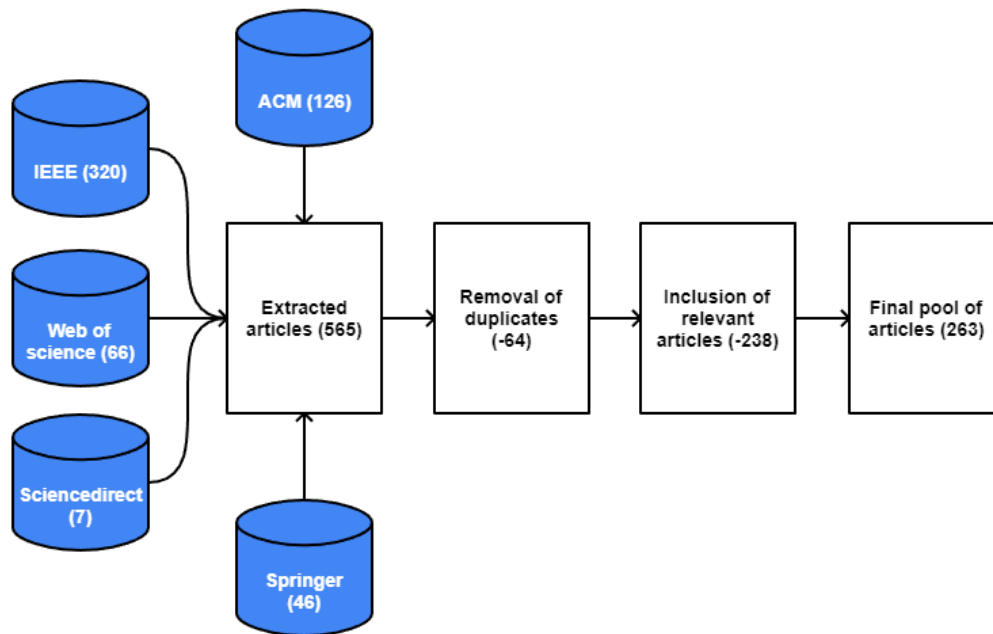[9]https://www.google.com/sheets/about
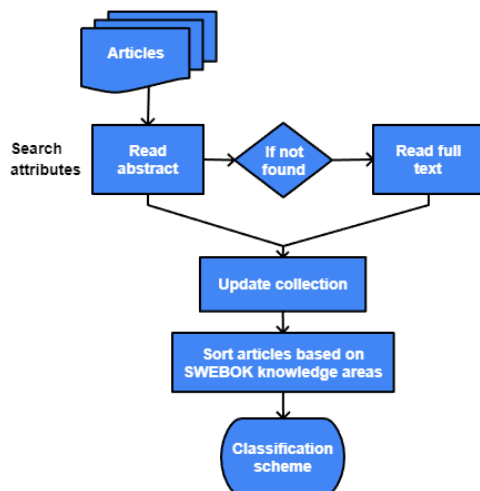
FIGURE 1. Article extraction process.

FIGURE 2. Attribute extraction leading to the classification scheme.

attribute extraction process leading to the resulting classification scheme is shown in Fig. 2.

V. STUDY FINDINGS
Here we answer the RQs, which we discussed in Section IV.

A. SDLC STAGES (Q1.1)
As already discussed, based on the analysis of articles, we have grouped them into 5 major categories (inspired by the aforementioned knowledge areas in SWEBOK [17]). These categories are briefly described as follows:

1) SOFTWARE REQUIREMENTS
We group all those articles in this category, which are concerned with the elicitation, modeling, analysis, prioritization, and validation of software requirements.

2) SOFTWARE ARCHITECTURE AND DESIGN
We group all those articles in this category, which deal with the process of specifying the architectural components and interfaces of software, and the description of how components of a software system are organized.

3) SOFTWARE IMPLEMENTATION
We group all those articles in this category, which are concerned with the development or construction of software achieved through a combination of design artifacts and coding.

4) SOFTWARE QUALITY ASSURANCE AND ANALYTIC
We group all those articles in this category, which deal with fundamental elements such as quality characteristics, quality process improvement or assessment, or verification and validation. We have also included articles referring to software testing in this category.

5) SOFTWARE MAINTENANCE
We group all those articles in this category, which deal with software adherence activities in order to meet new or changed operating environments such as refactoring, maintenance cost estimation, defect correctness, and factors related to software aging (e.g., resource depletion).
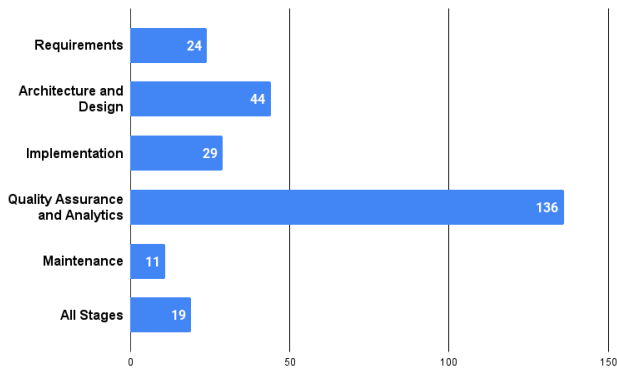
FIGURE 3. Articles by SDLC stages.

The SE stages and the number of articles that are associated with those stages are shown in Fig. 3. 136 out of 263 (52%) articles belong to quality assurance and analytic. 44 out of 263 (17%) articles have focused on architecture and design. 29 out of 263 (11%) articles have addressed the implementation followed by requirements engineering stage with 24 out of 263 (9%) articles. 11 (4%) articles were focusing on the maintenance phase. The rest of the articles were not particularly focusing on any stage but were generally applicable to SE.

B. APPLICATIONS OF ML FOR SE (Q1.2)

To address this question, we have developed a classification scheme based on the identified applications of ML for SE in order to characterize the obtained articles into appropriate categories. We have organized the applications of ML for SE as branches, which belong to five life cycle stages of SE (knowledge areas). The applications of ML for SE that come under corresponding SDLC stages along with the references of articles are shown in Table 1. Note that the applications highlighted in this study may not cover the entire knowledge base but rather should be deemed as stemming research indicating key applications of ML for SE in literature. The applications falling under the SE stages are described below.

1) APPLICATIONS OF ML AIMING AT SOFTWARE REQUIREMENTS

ML has been widely used to facilitate the software requirements stage. For instance, in requirements modeling and analysis, articles focused on distinguishing ambiguous requirements [21], resolving incompleteness, the correctness of requirements [22], etc. Requirements selection/prioritization/classification deals with articles proposing ML techniques that emphasize on automating prioritization of requirements or their classification. Perini et al. [23] employed an ML technique to generate approximate rank in order to prioritize requirements. Navarro-Almanza et al. [24] used a convolutional neural network (CNN) to classify functional requirements by analyzing textual features. We further found articles focusing on requirements traceability. Requirements traceability refers to

the ML approaches that assist in linking requirements to code or other artifacts as shown by Guo et al. [25], who used deep learning (DL) techniques in order to generate a trace link of requirements with other artifacts.

2) APPLICATIONS OF ML AIMING AT SOFTWARE ARCHITECTURE AND DESIGN

Many types of research in the past have applied ML to software architecture and design. The applications include design models, which are comprised of recommendation models for software processes/services. Apart from this, model smells and refactoring techniques of object-oriented structures using ML have also been proposed in the articles. White et al. [26] introduced DL to software language modeling based on information retrieval models. Design pattern prediction primarily focuses on recognizing design patterns in software through source code or user interface layout using ML techniques. For example, Nguyen et al. [27] proposed an approach known as DeepUI in order to semi-automate the design tasks by learning from previous UI design patterns. Development effort estimation refers to estimating effort for the development of software projects using ML techniques. Ionescu [28] used ANN to automate effort estimation by learning from textual features of project tasks.

3) APPLICATIONS OF ML AIMING AT SOFTWARE IMPLEMENTATION

We found several studies on ML assisting the software implementation stage. Among many applications, code clone/localization/refactoring/ labeling aims at finding code duplication, specific location of code in software, refactoring of code, or labeling of code with the help of ML, e.g., Alahmadi et al. [29] employed CNN in order to predict the code blocks in video tutorials. Code/bad smell detection focuses on applying ML in order to detect code and bad smells in software source code and design models, respectively. Code smells are indications of poor software code quality leading to the rise of technical debt. It generally includes god classes, spaghetti code, etc., whereas bad smells in design models have similar characteristics such as lazy classes and middle man. Pecorelli et al. [30] investigated data balancing techniques and addressed unbalanced dataset issues when employing ML for code smell detection. Maneerat et al. [31] proposed an approach to predict bad smells from design models such as class diagrams. Code inspection/analysis represents the class in which an ML technique is employed for the purpose of code reviews. For instance, Lal et al. [32] proposed an ML approach to automate code reviews for the pushed code. The code/program similarity category refers to the identification of specific piece(s) of code, which are similar between two or more software projects. Additionally, Kim et al. [33] proposed an ML technique in order to reduce the number of program similarity comparisons aimed at distinguishing between original and pirated/cracked software.

4) APPLICATIONS OF ML AIMING AT SOFTWARE QUALITY AND ANALYTIC

Most of the articles we found were focusing on applying ML to various software quality assurance and analytic tasks. The applications include: fault/bug/defect prediction category, which revolves around the prediction of faults, bugs, or defects using ML techniques [34]–[39]. Test case/ data/oracle generation surrounds ML techniques that help in generating test data, test oracles, or entire test suites. Braga *et al.* [40] proposed an ML technique to automate the process of test oracle generation. Test case selection/ prioritization/ classification deals with the class that particularly focuses on test case prioritization or classification techniques using ML. Rosenfeld *et al.* [41] employed an ML technique in order to select generic test cases for android applications. The technique is aimed at reducing the manual testing efforts by classifying the activities and automatically selecting the activity-specific test cases. Vulnerability/ anomaly/malware discovery/analysis mostly concerns the security aspect of the software, e.g., Huang *et al.* [42] employed the term frequency-inverse document frequency (TF-IDF) technique and deep neural network to automatically classify software vulnerabilities. Software analysis, technique assessment, and software process assessment come under assessment and analysis of software. In this regard, Fu *et al.* [43] proposed a regression-based ML technique in order to estimate software energy consumption by analyzing software performance features. The verification and validation category specifically addresses prediction and verification of software reliability through ML, e.g., Tamura *et al.* [44] proposed a DL-based technique to select the most suitable software reliability model for the development project. Testing effort estimation refers to the amount of testing effort required in order to test a software system using ML techniques, e.g., Silva *et al.* [6] evaluated various ML tools in order to estimate the execution times for running functional test cases.

5) APPLICATIONS OF ML AIMING AT SOFTWARE MAINTENANCE

The software maintenance stage has been found as the least focused stage for researchers in this domain. In this category, the research is more inclined towards cost/effort estimation than the rest of the maintenance tasks. We found articles focusing on software maintainability prediction, which refers to the proposed ML techniques in order to assist the prediction of maintainability metrics appropriate for specific software projects [45]. Software aging detection refers to the use of ML in order to detect software maturity and its aging in terms of resource depletion such as memory leaks, high CPU usage, and overtime. In this regard, Andrzejak *et al.* [46] investigated the feasibility of ML techniques for classification in detecting early performance degradation due to software image aging. The maintenance effort estimation class aims at estimating the amount of

effort required for the maintenance of a software system using ML, e.g., Chandra *et al.* [47] used an SVM-based regression model in order to forecast maintenance effort with univariate and multivariate approaches.

6) EFFECT AND SIGNIFICANCE OF APPLYING ML AT EACH SDLC STAGE

ML aims to automate and support the SE activities, which are considered to be performed intensively by humans. ML allows systems to perform human-centric activities at a much larger scale [48]. In fact, an empirical study [49] has been conducted to understand whether software engineers can utilize ML techniques for the improvement of their SE process and whether solutions proposed by engineers still outperform ML techniques. However, the need for ML techniques is still pertinent due to their ability to outperform in most SE activities. We highlight some of these activities with respect to SDLC stages which are as follows:

In the requirements stage, writing requirements specifications is highly deemed to be a human-centric task. Prior work by Pandita *et al.* [50] and Jahan *et al.* [51] have inferred the most probable specifications and identified its unexpected behaviors from various artifacts by employing ML techniques, respectively. Ferrari *et al.* [52] identified ambiguous requirements among different domains using ML. In the architecture and design stage, predicting design patterns is an important reverse engineering activity to improve software integrity. However, it often suffers from false positives and negatives [53]. As the number of patterns is increasing rapidly due to their variations, the process of recognizing these patterns can be effectively learned using ML [53]. In the implementation stage, detecting code smells in a large codebase can be extremely difficult for a human as opposed to a machine, thus ML techniques can greatly reduce this effort of detecting code smells or technical debt [30], [31]. In quality assurance, there is a need to ensure that the system remains error-free or to be able to timely identify the cause of failure. ML techniques employed in literature for this purpose proved to be promising in detecting software faults [34]–[36]. Test generation is also considered to be a task that requires human intelligence. Zhang *et al.* [54] have employed ML to automatically generate test data in order to improve return on investment. In software maintenance, Malgonde *et al.* [55] have shown ML techniques perform significantly better at predicting the effort as compared to the team estimates (human-centric).

Despite the intriguing tendency of full automation, complete automation could often result in a potentially fallible system, therefore, practitioners are encouraged to employ ML techniques with humans in the loop wherever there is a presence of criticality [1], [49]. In addition, there is a significant lack of studies showing the cost-benefit analysis of their proposed ML techniques, which would be vital for ML-based approaches to be feasible for adaptation in the industry.

TABLE 1. Classification by articles.

SDLC Stages	Applications of ML for SE	Articles
All Stages	N/A	[1, 11, 14, 48, 49, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69]
Requirements	Requirements Modeling and Analysis	[21, 22, 51, 70, 71, 72, 73, 74, 75]
	Requirements Selection/Prioritization/Classification	[23, 24, 76, 77, 78, 79, 80, 81]
	Requirements Traceability	[3, 25, 82, 83, 84, 85, 86]
Architecture and Design	Design Modeling	[9, 26, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102]
	Design Pattern Prediction	[27, 53, 103, 104, 105, 106]
	Development Effort Estimation	[4, 5, 28, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122, 123]
Implementation	Code Clone/Localization/Refactoring/Labeling	[29, 124, 125, 126, 127, 128, 129, 130, 131, 132]
	Code/Bad smell detection	[30, 31, 133, 134]
	Code Inspection/Analysis	[32, 135, 136, 137, 138, 139, 140, 141, 142]
	Code/Program Similarity	[33, 143, 144, 145, 146, 147]
Quality Assurance and Analytic	Fault/Bug/Defect Prediction	[7, 34, 35, 36, 37, 38, 39, 148, 149, 150, 151, 152, 153, 154, 155, 156, 157, 158, 159, 160, 161, 162, 163, 164, 165, 166, 167, 168, 169, 170, 171, 172, 173, 174, 175, 176, 177, 178, 179, 180, 181, 182, 183, 184, 185, 186, 187, 188, 189, 190, 191, 192, 193, 194, 195, 196, 197, 198, 199, 200, 201, 202, 203]
	Test Case/Data/Oracle Generation	[40, 54, 204, 205, 206, 207, 208, 209]
	Test Case Selection/Prioritization/Classification	[41, 210, 211, 212, 213]
	Vulnerability/Anomaly/Malware Discovery/Analysis	[42, 214, 215, 216, 217, 218, 219, 220, 221, 222, 223, 224, 225, 226, 227, 228, 229, 230, 231, 232]
	Software Analysis	[43, 233, 234, 235, 236, 237, 238, 239, 240, 241, 242, 243]
	Technique Assessment	[244, 245, 246, 247, 248]
	Software Process Assessment	[249, 250, 251]
	Verification and Validation	[44, 246, 252, 253, 254, 255, 256, 257, 258, 259, 260, 261, 262, 263, 264, 265]
	Testing Effort Estimation	[6, 266, 267, 268]
Maintenance	Software Maintainability Prediction	[45, 269, 270, 271]
	Software Aging Detection	[46, 272, 273, 274, 275]
	Maintenance Effort Estimation	[47, 276]

C. ML TYPE AND TECHNIQUES (Q1.3)

1) ML TYPES

By the type of ML, we mean how the models have been trained, i.e., supervised, semi-supervised, unsupervised, reinforcement, or analytical learning. Supervised learning is based on a training set and a test set taken from the dataset. The model training is done by taking multiple labeled samples from the train set. After the model is trained, its performance is evaluated using the test set. In semi-supervised learning, both labeled and unlabelled data are employed in order to train the model. The dataset is divided into unsupervised clusters as such. Then, the class information is obtained by learning the clustering outcomes [216]. Unsupervised learning requires no training dataset. For instance, in unsupervised learning for fault detection, software instances are usually grouped into clusters and each cluster is labeled as "Buggy" or "Correct". However, each cluster needs to be labeled manually by the individuals with expertise [198]. Reinforcement learning refers to unsupervised goal-oriented learning performed by an agent directly interacting with the environment. Analytical learning is aimed at generating solutions based on background knowledge and improving inference iteratively [253].

As shown in Fig. 4, 193 out of 263 (73%) articles employed supervised learning, 15 out of 263 (6%) articles employed unsupervised learning, 6 out of 263 (2%)
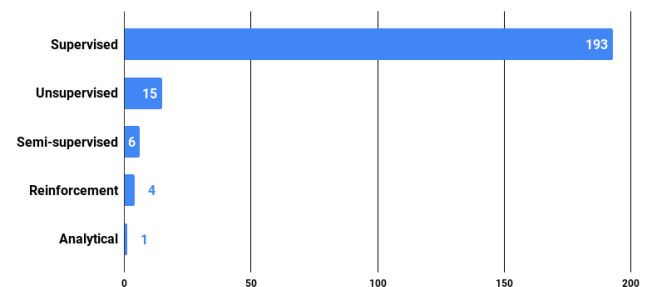
FIGURE 4. Articles by ML type.

articles employed semi-supervised learning, 4 out of 263 (2%) articles addressed reinforcement learning, and 1 out of 263 (0.4%) focused on analytical (inference-based) learning. The rest of the articles 44 out of 263 (17%) did not explicitly report the employed ML type.

2) ML TECHNIQUES

ML techniques are the algorithms used for classification, regression, or clustering problems, e.g., SVM, RF, or ANN. The employed techniques in the selected pool of articles are shown in Fig. 5. The topmost commonly used techniques are ANN, RF, DT, and NB, respectively. While 51 out of 263 (19%) articles employed ANN, 45 out of 263 (17%) articles
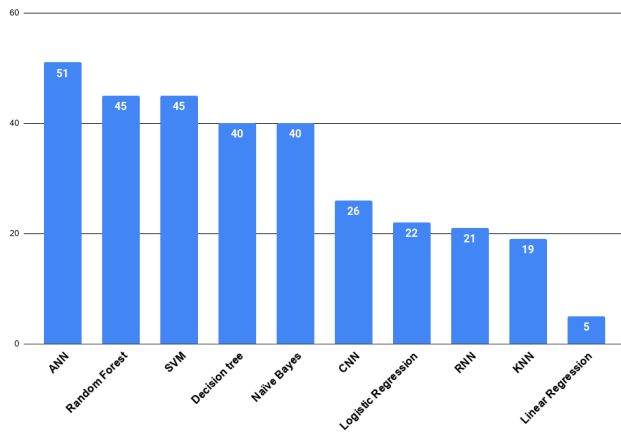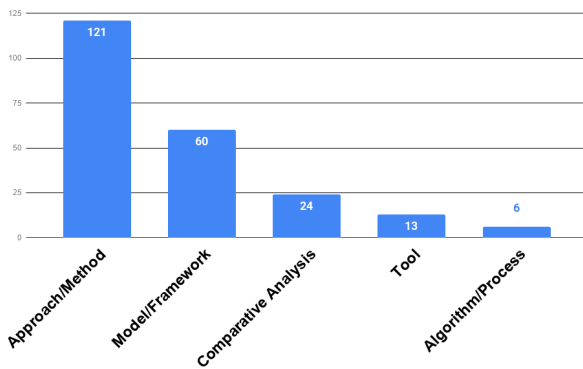
FIGURE 5. Articles by techniques.

FIGURE 6. Articles by contribution facet.

TABLE 2. Named propositions in the articles.

Sr. no.	Name	Contribution Facet	Article
1	Trace-by-Classification	Approach	[86]
2	DeepSim	Approach	[147]
3	CDGDroid	Approach	[200]
4	SLDeep	Approach	[175]
5	REMI	Approach	[168]
6	Feature Maps	Algorithm	[105]
7	ProbPoly	Framework	[75]
8	ExploitMeter	Framework	[231]
9	DLFuzz	Framework	[256]
10	DARVIZ	Framework	[98]
11	Seml	Framework	[172]
12	CroLSim	Model	[85]
13	DeepGauge	Process	[206]
14	WIRECAML	Tool	[224]
15	SOA-based integrated software	Tool	[151]
16	Modelware	Tool	[102]
17	Featuretools	Tool	[67]
18	Code-Buff	Tool	[128]
19	AppFlow	Tool	[211]
20	CloneCognition	Tool	[126]
21	ArchLearner	Tool	[95]
22	SZZ Unleashed	Tool	[149]
23	Auto-sklearn	Tool	[194]
24	RIVER	Tool	[209]
25	InSet	Tool	[91]
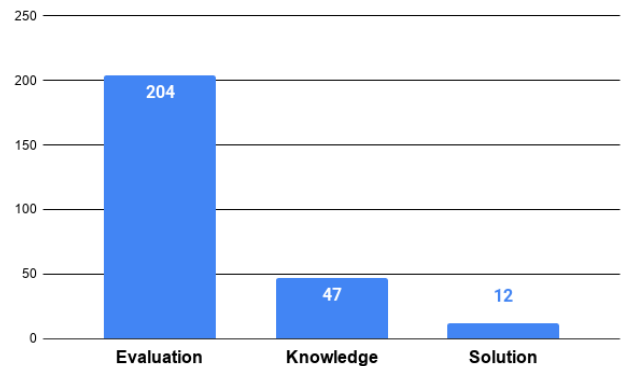
FIGURE 7. Articles by research facet.

have used RF and SVM, and 40 out of 263 (15%) articles used DT and NB for model training.

D. CONTRIBUTION FACET OF THE ARTICLES (Q2.1)

The contribution facet addresses the novel propositions of the articles. It is derived by analyzing the contribution of the articles, which represents the current state-of-the-art and enables researchers and industrial practitioners to get an overview of the existing tools and techniques in the literature. As shown in Fig. 6, 121 out of 263 (46%) articles focused on approaches/methods, followed by 60 (23%) articles proposing models/frameworks, 24 (9%) articles focusing on comparative analysis of existing techniques, 13 (5%) articles focusing on tools, and 6 (2%) articles focusing on algorithms/processes. The rest of the articles – 39 out of 263 (15%) – reported no new propositions. These articles were either investigating existing approaches, discussing opinions, or reporting their experiences.

Table 2 shows the names of the propositions along with the contribution facet and references of the articles. Interestingly, only 25 out of 263 (9%) articles have explicitly named their propositions.

E. RESEARCH FACET OF THE ARTICLES (Q2.2)

The research facet describes the nature of articles in terms of their purpose of conducting the research, such as evaluations (articles employing empirical methods such as controlled experiments or case studies), solutions (articles proposing solutions to underlying problems without empirical evidence), and knowledge (articles expressing experiences and opinions). Fig. 7 shows the articles by their research facet. 204 out of 263 (78%) articles have contributions with empirically evaluated propositions, whereas 47 out of 263 (18%) articles are knowledge-based, and 12 out of 263 (5%) articles have proposed solutions without any empirical evaluation.

The evaluation facet, in turn, represents the type of evaluation that has been performed in the articles in order to

FIGURE 8. Articles by evaluation facet.

FIGURE 9. Articles by year.

FIGURE 10. Articles by venues (Top 5).

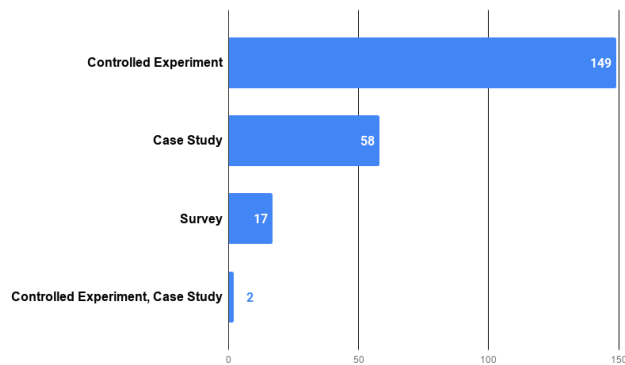evaluate their propositions. The articles by the evaluation facet are shown in Fig. 8. Controlled experiments have been performed in 148 out of 204 (73%) articles followed by case studies in 58 out of 204 (28%) articles and surveys in 16 out of 204 (8%) articles. 2 out of 204 (1%) articles have employed both a controlled experiment and a case study for an empirical evaluation; whereas, rest of the articles did not use any empirical method for evaluation purposes. Moreover, we found no article employing ethnography or action research as empirical methods for evaluation. Among the articles those performed control experiments, 78 articles proposed approaches/techniques/methods, and 41 articles proposed models/frameworks. While 15 articles focused on comparative analysis, 8 articles proposed tools, and 4 articles introduced new algorithms/processes.

F. DATASETS (Q2.3)
We further explored the datasets that have been used in most of the articles in order to evaluate their proposed approaches or comparative studies. Evidently, the majority of articles employed datasets obtained from PROMISE[10] repository followed by repositories made publicly available by NASA,[11] StackOverflow,[12] Github,[13] and JAVA projects.

G. TRENDS IN TERMS OF YEAR (Q3.1)
This refers to the trends in terms of publication years of articles. It shows the evolution of the adoption of ML for SE. As shown in Fig. 9, the use of ML for SE is consistently growing over the passage of time. One can also observe an exponential growth in this trend from 2016 - 2018, where 2018 proved to be the highest publication year with 63 (24%) publications. In 2019 and 2020, we recorded relatively fewer publications: 45 out of 263 (17%) and 34 out of 263 (13%), respectively. There could be two plausible reasons for that. Either some databases are not updated completely (as this study was conducted in Q4 of 2020) or like any hype cycle,
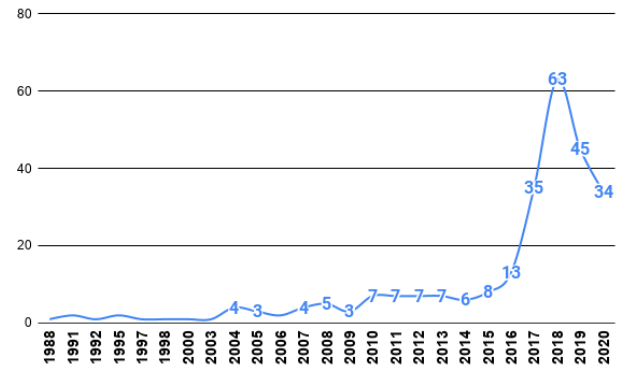
the peak of inflated expectations regarding ML for SE was reached in 2018 and now the trend is slowly going towards the trough of disillusionment. We believe the latter is the case here.

H. VENUES WITH HIGHEST PUBLICATIONS (Q3.2)
Fig. 10 shows the top 5 venues where most researchers of the domain tend to publish. International Conference on Software Engineering (ICSE) is leading by 11 out of 263 (4%) and the second most publishing venue is Transactions on Software Engineering (TSE) journal with 10 out of 263 (4%). They are followed by International Workshop on Machine Learning and Software Engineering, which featured 5 out of 263 (2%) articles, European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE), which also featured 5 out of 263 (2%) articles, and International Conference on Cloud Computing, Data Science & Engineering (Confluence), which featured 3 out of 263 (1%) articles. Moreover, Fig. 11 shows the overall distribution of articles with respect to publishing venues. Here one can observe that 155 out of 263 (59%) articles have been published in conferences, 51 out of 263 (19) articles have been published in journals, 26 out 263 (10%) articles have been published in workshops, and 18 out of 263 (7%) articles have been published in symposia.

VI. ANALYSIS AND DISCUSSION
This section relates to the fourth goal of this study (G4) and deals with implications and analysis of the aforementioned articles.

---

[10]http://promise.site.uottawa.ca/SERepository/datasets-page.html
[11]https://data.nasa.gov/
[12]https://archive.org/details/stackexchange
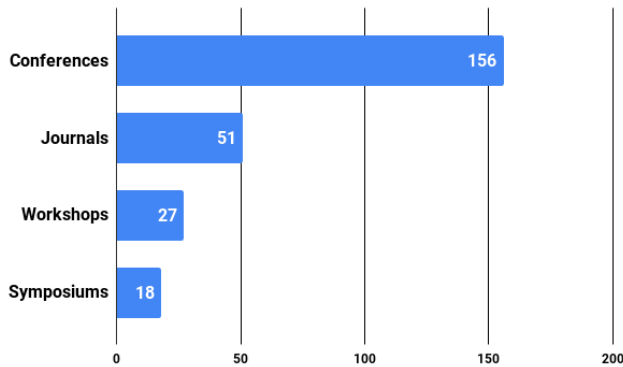[13]https://ghtorrent.org/

FIGURE 11. Articles by publishing venues.

A. RELATION OF SDLC STAGES WITH RESEARCH AND CONTRIBUTION FACETS

Fig. 12 shows the relationship of the contribution and research facets explored in this study with the SDLC stages. Moreover, the figure provides a bird's-eye view of the current studies falling into the respective SDLC stages along with their contribution type and research purpose. For instance, 55 articles belonging to the quality assurance stage have proposed a new approach or method as their primary contribution, and the contributions of 107 articles at this stage were evaluated empirically. In addition, we can observe that no tool has been proposed for the requirements and maintenance stage indicating less interest of researchers in prototyping their proposition.

B. RELATION OF SDLC STAGES WITH ML

As shown in Fig. 3, 52% of the articles were dedicated to the quality assurance and analytic stage, which shows that software quality[14] is the prime focus for the researchers of this domain. Indeed, quality assurance, along with requirements and design, are human-centric stages of the SDLC and the high number of articles in these areas highlight the fact that ML is able to offer help here. As shown in Tab. 1, we further observed that fault/bug/defect prediction has been the major focus of researchers within quality assurance. Certainly, the nature of ML types and techniques is more supportive for this kind of activities, but we hope that in the future other SE activities may also similarly benefit from ML. This is particularly valid for the maintenance stage, which has been the least interesting area for the application of ML. We encourage researchers to investigate how ML can be used to automate certain tasks in this area. We further encourage researchers to adopt combinations of ML techniques and use diverse datasets from different sources in order to train the ML models so that the applicability of the techniques can be generalized as also observed in [99], [115], [188], [237].

C. RELATION OF SDLC STAGES WITH ML TYPES

As shown in Fig. 4, a vast majority of articles falling into requirements, architecture and design, and implementation

---

[14]Our criteria for software quality assurance is shown in Tab. 1

categories are addressing the problems using supervised learning. For instance, [25] used supervised DL technique to identify trace links and predict associations within artifacts. A similar supervised learning technique has been proposed in [86] order to generate trace links from commonly occurring artifacts in the project. The reason supervised learning is mostly employed in the articles could be that supervised learning models are comparatively simple and produce results with high confidence and accuracy. We also noticed that only 4 out of 263 (2%) articles [3], [61], [225], [238] used reinforcement learning. This implies a little interest of researchers in the applications of reinforcement learning to SE. Reinforcement learning has proven to be beneficial in solving complex problems especially in healthcare, business, and robotics [277]. Thus, we believe it would be an interesting area to explore in terms of facilitating SE. For instance, software simulations can be deemed as an environment in which the RL agent can interact and reach various goal-oriented outcomes [278].

D. RELATION OF SDLC STAGES WITH ML TOOLS

As shown in Fig. 6, only 13 articles proposed a new tool to facilitate SDLC stages. As further can be observed in Fig. 12, 6 out of those 13 tools have been proposed for quality assurance purposes, e.g., the tool named "Appflow", which is proposed by Hu *et al.* [211] and predicts reusable UI test cases using neural networks. Tools are indeed a valuable contribution when it comes to the practicality and applicability of the proposed approach. In the future, more tools are desirable that are targeting other SDLC stages.

E. RELATION OF SDLC STAGES WITH ML TECHNIQUES

Although all ML techniques have certain pros and cons, the selection of the most suitable technique depends on the type of dataset being constructed or employed and what problem is being addressed. The SDLC stage-wise breakdown of ML techniques is shown in Fig. 13. As anticipated, mostly ML techniques were employed to solve problems related to the quality assurance and analytic stage. ANN was the most commonly used technique here (30 articles), followed by SVM (28 articles) and RF (24 articles), respectively. NB was next in line with 21 articles. ANN, which was used in 30 articles in the quality assurance stage was also a subject of interest for the researchers working in the architecture and design stage (15 articles).

As shown in Fig. 13, ANN is the most widely employed ML technique for SDLC stages in general due to its simplicity and strong classification and regression capabilities. CNN is mostly used in supervised learning problems, whereas RNN has been used to address both supervised and unsupervised learning problems. In traditional ML techniques, KNN, k-means clustering, NB, and SVM are mostly employed to address semi-supervised and unsupervised learning problems. In the case of reinforcement learning, Q-learning technique and its variants have been mostly employed in the literature.
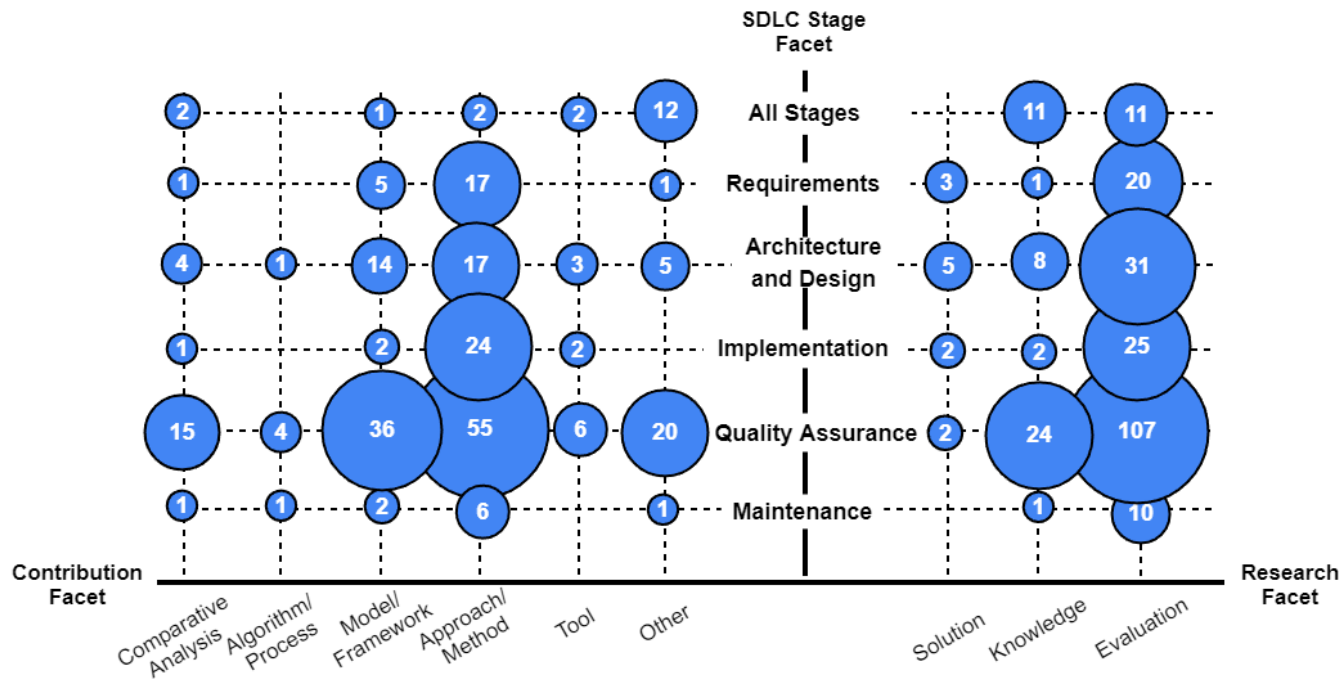
FIGURE 12. Relationship of contribution/research facets with the SDLC stage facet.
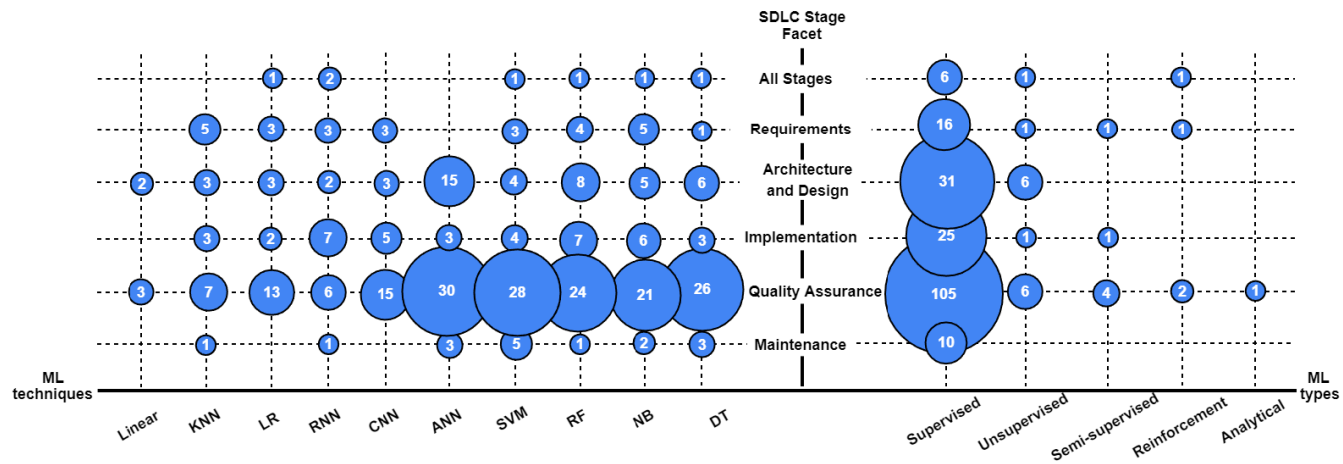
FIGURE 13. ML techniques usage in SDLC.

When it comes to neural networks-based techniques, our findings show that simple neural networks, e.g., ANN (51 out of 263 (19%)) and shallow neural networks, e.g., CNN and RNN (containing one or more hidden layers) (combined 47 out of 263 (18%)) are the most widely used ML techniques in SE. Neural networks are mostly employed for software architecture and design, and software implementation. Apart from neural networks, traditional ML techniques such as Boosting, NB, and case-based ranking, were popular in requirements engineering, particularly. The SVM technique has been mostly employed for the software maintenance stage. Apart from the ML techniques, most of the articles addressed problems related to supervised learning indicating classification as a major area of interest. While unsupervised and semi-supervised learning has been less employed in the area. The wide adoption of neural networks-based techniques in articles indicate their suitability and potential for achieving good results in this area. Mainly due to the reason that a neural network-based model is capable of learning from high dimensional large scale input data and an appropriate selection of cost function leads to the development of a more robust model. Moreover, neural network-based techniques are highly customizable and can be applied to various learning problems, such as supervised, unsupervised, or reinforcement, which make them highly flexible in terms of applicability.

Table 3 contains the complete list of articles (263) used in this paper showing ML techniques employed in those articles with respect to SDLC stages along with their contribution facet.

VII. CHALLENGES, LIMITATIONS, AND FUTURE RESEARCH DIRECTIONS

This section also relates to the fourth goal of this study (G4) and deals with challenges, limitations, and future research directions in this field.

A. CHALLENGES

One of the major challenges in this domain, as also reported by other experts, e.g., [150], [157], is the uncertain and stochastic nature of the employed ML techniques, and the difference in the captured data and results, e.g., the difference in the DL model output values when executing it multiple times over the same input data. The approaches need to be reproducible and rigorous in order to build high confidence for their application.

The availability of sufficiently labeled and structured datasets is also a challenge as also reported by other researchers, e.g., [32], [170], [184]. However, this can be overcome rather easily as more and more researchers have started sharing their datasets publicly. An associated issue is the imbalanced sizes of software projects and datasets. Using new techniques for dataset balancing, such as SMOTE and ClassBalancer (both evaluated by Percorelli *et al.* [30]), or devising new ones is highly recommended in this context.

The ever-increasing software complexity is also one of the greater challenges for this domain. Meinke *et al.* [63] also attest to our observation and further suggest that the scalability problem should be given proper attention by researchers of this domain. We also invite researchers to conduct more studies investigating the impact of ML techniques on different software sizes.

B. LIMITATIONS

As observed in some studies, e.g., [140], [176], the lack of generalizability regarded as over-fitting problems is one of few major limiting factors, which decreases the accuracy of results. This leads to lesser results when ML models are applied to diverse cross-project datasets. Using standard data preprocessing techniques such as SMOTE, ClassBalancer, and Resample [30], and performing K-fold cross-validation or hold-out validation could reduce the problem of over-fitted and under-fitted models.

As observed in some studies, e.g., Ghaffarian *et al.* [219], the current state of evaluation of ML techniques, especially for software vulnerability testing is not well grounded. The dataset often lacks sufficient vulnerability types, which results in less generalizable outcomes. In order to improve results' precision, lesser false positives, and false negatives while maintaining recall can help produce meaningful results.

In a distributed software development environment, manual inspection/allocation of work items, excessive time consumption, potentially fallible outcomes, and lack of production-ready approaches are some of the limitations identified by Barcus *et al.* [279] and Achimugu *et al.* [280].

C. FUTURE RESEARCH DIRECTIONS

In order to facilitate requirements traceability, researchers have suggested that devising a feedback mechanism, such as adding user feedback during the model training process in order to improve feature selection and performance, can really help the cause of generalizability. One of such works is presented by Sultanov *et al.* [3], which provides a very good basis for further developments.

In order to improve prediction accuracy and better reliability of results, more experiments using larger numbers of datasets and software applications have also been suggested [99], [115], [188], [237].

Researchers in the articles have also suggested investigating further regarding the suitable metrics and loss functions employed in the evaluation of ML for SE-focused techniques, especially for multi-class classification problems [125].

Future research directions also include automata learning for emergent middle-wares and using ML to address complex system integration problems, especially in system of systems such as the internet of things. Moreover, researchers are encouraged to devise adaptable, easily integrable, and scalable solutions in the area.

VIII. THREATS TO VALIDITY

Similar to other secondary studies, this study is also prone to some validity threats. The threats and their mitigation strategies are described in this section.

A. EXTERNAL VALIDITY

The extraction of articles and choice of repositories constitute a threat to internal validity. In order to minimize the former, we adopted the PICO (Population, Intervention, Comparison, Outcomes) criteria suggested by Petersen *et al.* [19] to formulate the search terms. The selected terms unequivocally represent the goals of our work. An associated issue corresponds to the frequently used specific ML terms. Although the query used did not explicitly include ML terms, such as classification, regression, SVM, ANN, inductive logic, Bayesian network, or deep belief network, this would not affect the analysis much because such information is usually available in abstracts, hence accessible. In order to minimize the latter, we used five digital libraries as the primary source for this research. All selected digital libraries are well known in the computer science discipline for including the most relevant results [281]. Additionally, according to Wohlin *et al.* [282], having a larger set of papers is not necessarily better for such reviews. The important thing is that the found studies are a good representation of the population, which we ensured in this study by adopting a rigorous paper selection process.

TABLE 3. Articles by ML techniques.

	Refs.	SDLC Stages	Contribution Facet	ML techniques
1	[58]	All Stages	Approach/Technique/Method	RNN
2	[63]	All Stages	Other	Other
3	[64]	All Stages	Other	Other
4	[56]	All Stages	Comparative Analysis	Other
5	[57]	All Stages	Other	Other
6	[69]	All Stages	Other	RNN, RBM
7	[14]	All Stages	Model/Framework	Other
8	[67]	All Stages	Tool	RF
9	[49]	All Stages	Other	Other
10	[62]	All Stages	Tool	NLP
11	[61]	All Stages	Other	DT
12	[48]	All Stages	Other	Other
13	[65]	All Stages	Other	Other
14	[1]	All Stages	Other	Other
15	[60]	All Stages	Other	Other
16	[68]	All Stages	Comparative Analysis	LR, SVM, NB
17	[66]	All Stages	Approach/Technique/Method	LSTM
18	[11]	All Stages	Other	Other
19	[59]	All Stages	Other	Other
20	[21]	Requirements	Approach/Technique/Method	NB, KNN, RF
21	[70]	Requirements	Approach/Technique/Method	SVM, SMO, NB
22	[82]	Requirements	Approach/Technique/Method	PN
23	[75]	Requirements	Model/Framework	ProbPoly
24	[71]	Requirements	Approach/Technique/Method	Text2Model
25	[84]	Requirements	Approach/Technique/Method	RF
26	[72]	Requirements	Approach/Technique/Method	Other
27	[22]	Requirements	Approach/Technique/Method	NB, RF, LR, SGD, DT
28	[23]	Requirements	Approach/Technique/Method	Boosting
29	[81]	Requirements	Approach/Technique/Method	NSGA-II algorithm
30	[24]	Requirements	Model/Framework	CNN
31	[73]	Requirements	Approach/Technique/Method	FL
32	[76]	Requirements	Approach/Technique/Method	LP, SMO, NB, KNN
33	[86]	Requirements	Approach/Technique/Method	J48, FSS, CFS
34	[25]	Requirements	Model/Framework	RNN
35	[85]	Requirements	Model/Framework	KNN
36	[79]	Requirements	Other	Other
37	[3]	Requirements	Approach/Technique/Method	RL
38	[80]	Requirements	Model/Framework	LSTM, GRU, CNN
39	[51]	Requirements	Approach/Technique/Method	LSTM
40	[74]	Requirements	Approach/Technique/Method	Spacy NLP model
41	[77]	Requirements	Comparative Analysis	LR, SVM, MNB, kNN
42	[83]	Requirements	Approach/Technique/Method	RNN
43	[78]	Requirements	Approach/Technique/Method	RNN, CNN, SVM, KNN, LR, NB, RF
44	[117]	Architecture and Design	Approach/Technique/Method	KNN, CTM, MARS, CART
45	[102]	Architecture and Design	Tool	Modelware
46	[98]	Architecture and Design	Model/Framework	DARVIZ
47	[96]	Architecture and Design	Approach/Technique/Method	RF
48	[97]	Architecture and Design	Model/Framework	Other
49	[87]	Architecture and Design	Model/Framework	CNN
50	[27]	Architecture and Design	Approach/Technique/Method	RNN, GAN
51	[89]	Architecture and Design	Approach/Technique/Method	SVM
52	[122]	Architecture and Design	Other	CBR, ANN, DT, BN, SVR, GA, AR
53	[112]	Architecture and Design	Comparative Analysis	CBR, ANN, CART
54	[101]	Architecture and Design	Approach/Technique/Method	NB, SMO, RF
55	[92]	Architecture and Design	Model/Framework	Restricted Boltzmann Machine
56	[99]	Architecture and Design	Model/Framework	GRBF
57	[53]	Architecture and Design	Approach/Technique/Method	RNN, DT
58	[111]	Architecture and Design	Model/Framework	ANN
59	[93]	Architecture and Design	Model/Framework	Other
60	[113]	Architecture and Design	Other	NN
61	[110]	Architecture and Design	Comparative Analysis	SVR
62	[115]	Architecture and Design	Approach/Technique/Method	NB
63	[90]	Architecture and Design	Model/Framework	NN
64	[116]	Architecture and Design	Model/Framework	GP, LMS, LR, MP, RBFN, SMO, AR, BAG, CR, DT, MSR, ZR, DS, RT
65	[94]	Architecture and Design	Model/Framework	CNN
66	[120]	Architecture and Design	Approach/Technique/Method	DT, NN
67	[119]	Architecture and Design	Comparative Analysis	GP, NN
68	[108]	Architecture and Design	Model/Framework	RBF, SVR, PCA

TABLE 3. *(Continued.)* Articles by ML techniques.

69	[4]	Architecture and Design	Approach/Technique/Method	RT, MLP, SVR
70	[5]	Architecture and Design	Comparative Analysis	ANN, RI, FL, CART, CBR
71	[26]	Architecture and Design	Other	NN
72	[114]	Architecture and Design	Approach/Technique/Method	ANN, SVM
73	[118]	Architecture and Design	Other	ANN, GA
74	[28]	Architecture and Design	Approach/Technique/Method	ANN
75	[121]	Architecture and Design	Other	ANN, GA
76	[100]	Architecture and Design	Approach/Technique/Method	Other
77	[109]	Architecture and Design	Approach/Technique/Method	NB, LR, RF
78	[105]	Architecture and Design	Algorithm/Process	CNN, RF
79	[123]	Architecture and Design	Model/Framework	DNN
80	[95]	Architecture and Design	Tool	LSTM
81	[106]	Architecture and Design	Approach/Technique/Method	SBL
82	[107]	Architecture and Design	Model/Framework	RF
83	[88]	Architecture and Design	Approach/Technique/Method	LR, NB, DT, RF, KNN
84	[9]	Architecture and Design	Model/Framework	k-means clustering
85	[104]	Architecture and Design	Approach/Technique/Method	Other
86	[103]	Architecture and Design	Approach/Technique/Method	ANN, SVM, RF
87	[91]	Architecture and Design	Tool	NB, NN, KNN, RF, SVM, DT
88	[131]	Implementation	Approach/Technique/Method	RNN
89	[29]	Implementation	Approach/Technique/Method	CNN
90	[128]	Implementation	Tool	KNN
91	[132]	Implementation	Approach/Technique/Method	Fica
92	[33]	Implementation	Approach/Technique/Method	NN, RF
93	[125]	Implementation	Model/Framework	CNN
94	[145]	Implementation	Approach/Technique/Method	RNN
95	[127]	Implementation	Approach/Technique/Method	CNN
96	[147]	Implementation	Approach/Technique/Method	DNN
97	[133]	Implementation	Approach/Technique/Method	DT
98	[138]	Implementation	Approach/Technique/Method	OGUST
99	[32]	Implementation	Approach/Technique/Method	NB, DT, SVM
100	[140]	Implementation	Approach/Technique/Method	RF, NB, KNN
101	[31]	Implementation	Approach/Technique/Method	RF, NB, LR
102	[137]	Implementation	Comparative Analysis	NB
103	[146]	Implementation	Model/Framework	RNN
104	[143]	Implementation	Approach/Technique/Method	CNN, RNN, LSTM
105	[135]	Implementation	Approach/Technique/Method	SVM
106	[126]	Implementation	Tool	ANN
107	[30]	Implementation	Approach/Technique/Method	Other
108	[124]	Implementation	Approach/Technique/Method	LSTM
109	[141]	Implementation	Approach/Technique/Method	RF, J48, SMO, MLP, NB, LogitBoost, Ad-aBoost
110	[134]	Implementation	Approach/Technique/Method	DT, GBT, SVM, RF, ANN
111	[142]	Implementation	Approach/Technique/Method	RNN
112	[129]	Implementation	Approach/Technique/Method	RNN
113	[130]	Implementation	Approach/Technique/Method	KNN, RF
114	[136]	Implementation	Approach/Technique/Method	DNN
115	[139]	Implementation	Approach/Technique/Method	NB, LR, SVM, RF, XGB, CNN
116	[144]	Implementation	Approach/Technique/Method	RNN
117	[150]	Quality Assurance and Analytic	Approach/Technique/Method	SVM, DT
118	[267]	Quality Assurance and Analytic	Approach/Technique/Method	COBWEB/3
119	[224]	Quality Assurance and Analytic	Tool	DT, RF, LR, NB, TAN
120	[256]	Quality Assurance and Analytic	Model/Framework	CNN
121	[161]	Quality Assurance and Analytic	Approach/Technique/Method	OC-SVM
122	[219]	Quality Assurance and Analytic	Other	Other
123	[41]	Quality Assurance and Analytic	Approach/Technique/Method	KStar
124	[163]	Quality Assurance and Analytic	Approach/Technique/Method	NN
125	[266]	Quality Assurance and Analytic	Other	DT
126	[211]	Quality Assurance and Analytic	Tool	NN
127	[228]	Quality Assurance and Analytic	Model/Framework	RF, NB
128	[40]	Quality Assurance and Analytic	Approach/Technique/Method	AdaBoostM1, JRIP 3
129	[207]	Quality Assurance and Analytic	Approach/Technique/Method	ANN, DT, KNN, NB, RF, SVM
130	[254]	Quality Assurance and Analytic	Model/Framework	RNN
131	[206]	Quality Assurance and Analytic	Algorithm/Process	DNN
132	[255]	Quality Assurance and Analytic	Approach/Technique/Method	LSTM
133	[157]	Quality Assurance and Analytic	Approach/Technique/Method	SVM, CNN
134	[265]	Quality Assurance and Analytic	Model/Framework	RNN
135	[210]	Quality Assurance and Analytic	Algorithm/Process	SVM
136	[54]	Quality Assurance and Analytic	Model/Framework	GA
137	[253]	Quality Assurance and Analytic	Model/Framework	EDAs
138	[262]	Quality Assurance and Analytic	Model/Framework	MBR, BBN

TABLE 3. *(Continued.)* Articles by ML techniques.

139	[205]	Quality Assurance and Analytic	Other	Other
140	[212]	Quality Assurance and Analytic	Approach/Technique/Method	K-means clustering, Expectation–Maximization, Incremental Conceptual Clustering
141	[252]	Quality Assurance and Analytic	Other	Other
142	[246]	Quality Assurance and Analytic	Comparative Analysis	DT, BNN, RBNN, SVM
143	[44]	Quality Assurance and Analytic	Approach/Technique/Method	NN
144	[223]	Quality Assurance and Analytic	Model/Framework	Other
145	[257]	Quality Assurance and Analytic	Approach/Technique/Method	SVM
146	[233]	Quality Assurance and Analytic	Other	Other
147	[170]	Quality Assurance and Analytic	Approach/Technique/Method	NB, DT, SVM
148	[182]	Quality Assurance and Analytic	Comparative Analysis	ANN, Particle Swarm Optimization, DT, NB
149	[151]	Quality Assurance and Analytic	Tool	SVM, DT
150	[268]	Quality Assurance and Analytic	Model/Framework	ANN
151	[264]	Quality Assurance and Analytic	Other	STP, LTP
152	[220]	Quality Assurance and Analytic	Approach/Technique/Method	DT, RF, KNN , SVM
153	[213]	Quality Assurance and Analytic	Approach/Technique/Method	NB
154	[204]	Quality Assurance and Analytic	Algorithm/Process	GA
155	[200]	Quality Assurance and Analytic	Approach/Technique/Method	CNN
156	[208]	Quality Assurance and Analytic	Approach/Technique/Method	Evolutionary Algorithm
157	[158]	Quality Assurance and Analytic	Approach/Technique/Method	DT
158	[188]	Quality Assurance and Analytic	Comparative Analysis	LR, ANN
159	[173]	Quality Assurance and Analytic	Comparative Analysis	NB
160	[176]	Quality Assurance and Analytic	Approach/Technique/Method	LR
161	[187]	Quality Assurance and Analytic	Approach/Technique/Method	ANN
162	[258]	Quality Assurance and Analytic	Other	NN
163	[229]	Quality Assurance and Analytic	Model/Framework	NN
164	[218]	Quality Assurance and Analytic	Model/Framework	RF, PART
165	[245]	Quality Assurance and Analytic	Comparative Analysis	NN, NB
166	[261]	Quality Assurance and Analytic	Other	Other
167	[263]	Quality Assurance and Analytic	Model/Framework	ANN
168	[167]	Quality Assurance and Analytic	Other	SVM, RF
169	[244]	Quality Assurance and Analytic	Approach/Technique/Method	SBL
170	[184]	Quality Assurance and Analytic	Model/Framework	DT, SVM, ANN
171	[153]	Quality Assurance and Analytic	Comparative Analysis	LM, MAE, LR, PR, SVR, NNC, SVLR, NND, LoR, NB, IBL, JDT, 1R
172	[152]	Quality Assurance and Analytic	Model/Framework	DT, MLP, RBF
173	[6]	Quality Assurance and Analytic	Comparative Analysis	SVR, ANN
174	[246]	Quality Assurance and Analytic	Comparative Analysis	DT, SVM
175	[249]	Quality Assurance and Analytic	Approach/Technique/Method	C4.5, NB, SVM
176	[214]	Quality Assurance and Analytic	Comparative Analysis	DT, NB, SVM-C, KNN, RF
177	[247]	Quality Assurance and Analytic	Approach/Technique/Method	SVM
178	[160]	Quality Assurance and Analytic	Other	Other
179	[251]	Quality Assurance and Analytic	Approach/Technique/Method	NN
180	[250]	Quality Assurance and Analytic	Model/Framework	NN
181	[216]	Quality Assurance and Analytic	Model/Framework	SVM
182	[186]	Quality Assurance and Analytic	Other	DT, CBR, ANN, SVM
183	[217]	Quality Assurance and Analytic	Approach/Technique/Method	Recurrent Neural Network, LSTM
184	[248]	Quality Assurance and Analytic	Other	NN
185	[178]	Quality Assurance and Analytic	Model/Framework	CNN
186	[177]	Quality Assurance and Analytic	Approach/Technique/Method	CNN
187	[165]	Quality Assurance and Analytic	Approach/Technique/Method	SVM, RNN
188	[171]	Quality Assurance and Analytic	Model/Framework	CNN
189	[231]	Quality Assurance and Analytic	Model/Framework	FL
190	[237]	Quality Assurance and Analytic	Model/Framework	SVM
191	[230]	Quality Assurance and Analytic	Approach/Technique/Method	CNN
192	[189]	Quality Assurance and Analytic	Comparative Analysis	MLP, RBF, CART, KNN
193	[221]	Quality Assurance and Analytic	Approach/Technique/Method	CNN
194	[174]	Quality Assurance and Analytic	Other	Single Layer Perceptron, Multi Layer Perceptron, LVQ, SOM, AIRS, CLONAL, Immune RF, DT, SVM, NB, LR
195	[193]	Quality Assurance and Analytic	Other	LSTM, NB, RF
196	[222]	Quality Assurance and Analytic	Approach/Technique/Method	DBN
197	[197]	Quality Assurance and Analytic	Approach/Technique/Method	CNN
198	[162]	Quality Assurance and Analytic	Model/Framework	ANN, CNN, SOM, LVQ, LVQ
199	[38]	Quality Assurance and Analytic	Comparative Analysis	Linear Regression, Ridge, Lasso, Random Forest Regression
200	[43]	Quality Assurance and Analytic	Model/Framework	Other
201	[259]	Quality Assurance and Analytic	Other	DT, LR
202	[34]	Quality Assurance and Analytic	Approach/Technique/Method	SVM
203	[236]	Quality Assurance and Analytic	Approach/Technique/Method	SVM

TABLE 3. *(Continued.)* **Articles by ML techniques.**

204	[198]	Quality Assurance and Analytic	Approach/Technique/Method	RNN
205	[155]	Quality Assurance and Analytic	Comparative Analysis	DNN
206	[202]	Quality Assurance and Analytic	Model/Framework	RNN
207	[35]	Quality Assurance and Analytic	Comparative Analysis	LR, NB, DT, J48
208	[37]	Quality Assurance and Analytic	Approach/Technique/Method	DT, RF, NB, SVM, ANN
209	[234]	Quality Assurance and Analytic	Other	NN, RF, DT
210	[203]	Quality Assurance and Analytic	Model/Framework	SDNN
211	[164]	Quality Assurance and Analytic	Comparative Analysis	GMMs, ANN
212	[180]	Quality Assurance and Analytic	Other	CNN
213	[242]	Quality Assurance and Analytic	Approach/Technique/Method	DT, KNN, SVM, NB
214	[149]	Quality Assurance and Analytic	Tool	RF
215	[239]	Quality Assurance and Analytic	Approach/Technique/Method	SGD
216	[227]	Quality Assurance and Analytic	Approach/Technique/Method	LSTM
217	[175]	Quality Assurance and Analytic	Approach/Technique/Method	LSTM
218	[39]	Quality Assurance and Analytic	Other	Other
219	[192]	Quality Assurance and Analytic	Approach/Technique/Method	ANN
220	[226]	Quality Assurance and Analytic	Approach/Technique/Method	RF, NB, J48
221	[201]	Quality Assurance and Analytic	Model/Framework	LSTM
222	[241]	Quality Assurance and Analytic	Model/Framework	LSTM
223	[36]	Quality Assurance and Analytic	Approach/Technique/Method	SVM
224	[191]	Quality Assurance and Analytic	Model/Framework	NaN
225	[185]	Quality Assurance and Analytic	Approach/Technique/Method	RF
226	[156]	Quality Assurance and Analytic	Approach/Technique/Method	LSTM
227	[169]	Quality Assurance and Analytic	Approach/Technique/Method	GA
228	[42]	Quality Assurance and Analytic	Model/Framework	TF-IDF, IG, DNN
229	[215]	Quality Assurance and Analytic	Model/Framework	LSTM
230	[240]	Quality Assurance and Analytic	Model/Framework	RFCM, LR, CART, KNN
231	[225]	Quality Assurance and Analytic	Algorithm/Process	RL
232	[194]	Quality Assurance and Analytic	Tool	RF, DT
233	[179]	Quality Assurance and Analytic	Comparative Analysis	LR, KNN, DT, RF, SVM, NN
234	[7]	Quality Assurance and Analytic	Approach/Technique/Method	RF
235	[172]	Quality Assurance and Analytic	Model/Framework	LSTM
236	[209]	Quality Assurance and Analytic	Tool	RNN
237	[238]	Quality Assurance and Analytic	Model/Framework	RL
238	[159]	Quality Assurance and Analytic	Model/Framework	LR, DT, RF
239	[235]	Quality Assurance and Analytic	Other	CART, kNN, KRR, MR, RF, SVR
240	[243]	Quality Assurance and Analytic	Other	Other
241	[196]	Quality Assurance and Analytic	Approach/Technique/Method	NHANES dataset
242	[190]	Quality Assurance and Analytic	Approach/Technique/Method	SVM, RF, ANN, DT, NBG, LR, CNN
243	[195]	Quality Assurance and Analytic	Approach/Technique/Method	NN
244	[232]	Quality Assurance and Analytic	Approach/Technique/Method	ANN
245	[154]	Quality Assurance and Analytic	Approach/Technique/Method	CNN
246	[260]	Quality Assurance and Analytic	Model/Framework	SVM, ANN, NB
247	[183]	Quality Assurance and Analytic	Approach/Technique/Method	RF, NB, SVM, ANN
248	[166]	Quality Assurance and Analytic	Model/Framework	RF, NB, DT, LR, ANN
249	[181]	Quality Assurance and Analytic	Approach/Technique/Method	MLP, CNN
250	[148]	Quality Assurance and Analytic	Model/Framework	NB, LR, C4.5, SVM, RF, MLP
251	[199]	Quality Assurance and Analytic	Approach/Technique/Method	CNN
252	[168]	Quality Assurance and Analytic	Approach/Technique/Method	RF
253	[274]	Maintenance	Model/Framework	SVM
254	[46]	Maintenance	Approach/Technique/Method	NB, SMO
255	[269]	Maintenance	Algorithm/Process	FL
256	[47]	Maintenance	Approach/Technique/Method	SVM
257	[272]	Maintenance	Approach/Technique/Method	M5P
258	[275]	Maintenance	Approach/Technique/Method	DT, ANN, SVM
259	[273]	Maintenance	Comparative Analysis	DT, SVM, DBN
260	[45]	Maintenance	Model/Framework	LSTM
261	[271]	Maintenance	Approach/Technique/Method	RF, NB, KNN, SVM, ANN
262	[270]	Maintenance	Other	ANN, SVM/R, DT
263	[276]	Maintenance	Approach/Technique/Method	RNN

B. INTERNAL VALIDITY

Another threat is regarding the quality assessment of this study. As discussed by Petersen *et al.* [283] and Kitchenham *et al.* [284], quality assessment is not common in such kind of studies as their overall aim is to give a broad overview of the topic area. However, despite these observations, we have adopted a rigorous process for the inclusion

and classification of papers, which ensures that only high-quality related papers are selected as primary studies.

C. CONCLUSION VALIDITY

Each article in this study was reviewed by the first author, which may lead to a threat to the reliability of the results. This threat was reduced by double-checking the article by the

second, the third, and the fourth author. A random set of articles was distributed among the second and the third author. Their review results were then compared with the results of the first author. In case of a disagreement, the opinion of the fourth author was sought. Although this did not happen much.

IX. CONCLUSION

The conclusion of the study is manifold. We have provided an overview of the state-of-the-art in the area of machine learning for software engineering by evaluating carefully selected studies. We also proposed a classification scheme that highlights the overall applications of machine learning for software engineering in terms of SDLC stages. The classification shows the primary focus of researchers towards specific stages. This observation is one of the major contributions of this study. This study also reveals that the quality of primary studies in the domain of ML and SE is evidence-based with respect to the techniques being empirically evaluated by the researchers. We have also shown the relationship of SDLC stages with ML types, tools, and techniques. Although this research area is showing moderate growth in terms of the number of publications, further primary studies need to be conducted to emphasize other lesser explored SDLC stages such as maintenance. The challenges, limitations and future directions reported in this article should motivate and further guide researchers in the future. We believe this study provides the necessary impetus and further motivation to explore those SDLC stages, which have been given lesser attention to date with respect to the application of ML.

In the future, we intend to perform a more comprehensive study investigating the relationship between ML and SDLC stages. To this end, we intend to narrow down our search query by including ML terms such as classification, regression, SVM, ANN, inductive logic, Bayesian network, or deep belief network. We believe in this way, we can grasp a more focused view of the state-of-the-art.

ACKNOWLEDGMENT

REFERENCES

[1] M. Harman, "The role of artificial intelligence in software engineering," in *Proc. 1st Int. Workshop Realizing AI Synergies Softw. Eng. (RAISE)*, Jun. 2012, pp. 1–6.

[2] D. Zhang, "Machine learning and software development," in *Proc. 14th IEEE Int. Conf. Tools With Artif. Intell.*, May 2010, pp. 87–119.

[3] H. Sultanov and J. H. Hayes, "Application of reinforcement learning to requirements engineering: Requirements tracing," in *Proc. 21st IEEE Int. Requirements Eng. Conf. (RE)*, Jul. 2013, pp. 52–61.

[4] P. L. Braga, A. L. I. Oliveira, and S. R. L. Meira, "Software effort estimation using machine learning techniques with robust confidence intervals," in *Proc. 19th IEEE Int. Conf. Tools With Artif. Intell. (ICTAI)*, Oct. 2007, pp. 352–357.

[5] J. J. Cuadrado-Gallego, P. Rodríguez-Soria, and B. Martín-Herrera, "Analogies and differences between machine learning and expert based software project effort estimation," in *Proc. 11th ACIS Int. Conf. Softw. Eng., Artif. Intell., Netw. Parallel/Distrib. Comput.*, Jun. 2010, pp. 269–276.

[6] D. G. E. Silva, M. Jino, and B. T. D. Abreu, "Machine learning methods and asymmetric cost function to estimate execution effort of software testing," in *Proc. 3rd Int. Conf. Softw. Test., Verification Validation*, 2010, pp. 275–284.

[7] H. D. Tran, L. T. M. Hanh, and N. T. Binh, "Combining feature selection, feature learning and ensemble learning for software fault prediction," in *Proc. 11th Int. Conf. Knowl. Syst. Eng. (KSE)*, Oct. 2019, pp. 1–8.

[8] V. H. S. Durelli, R. S. Durelli, S. S. Borges, A. T. Endo, M. M. Eler, D. R. C. Dias, M. P. Guimar, and M. P. Guimaraes, "Machine learning applied to software testing: A systematic mapping study," *IEEE Trans. Rel.*, vol. 68, no. 3, pp. 1189–1212, Sep. 2019.

[9] C. V. Bindewald, W. M. Freire, A. M. M. M. Amaral, and T. E. Colanzi, "Supporting user preferences in search-based product line architecture design using machine learning," in *Proc. 14th Brazilian Symp. Softw. Compon., Archit., Reuse*, Oct. 2020, pp. 11–20.

[10] S. Fajardo, R. F. García-Galvan, V. Barranco, J. C. Galvan, and S. F. Batlle, *Data Mining and Machine Learning for Software Engineering*, vol. 1. Rijeka, Croatia: InTech, 2016, p. 13. [Online]. Available: https://www.intechopen.com/books/advanced-biometric-technologies/liveness-detection-in-biometrics

[11] Z. Wan, X. Xia, D. Lo, and G. C. Murphy, "How does machine learning change software development practices?" *IEEE Trans. Softw. Eng.*, vol. 47, no. 9, pp. 1857–1871, Sep. 2020.

[12] D. Zhang, "Machine learning and software engineering," *Softw. Qual. J.*, vol. 11, no. 3, pp. 87–119, 2003.

[13] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *Proc. 3rd Int. Conf. Learn. Represent. (ICLR) Conf. Track*, 2015, pp. 1–14.

[14] F. Falcini, G. Lami, and A. M. Costanza, "Deep learning in automotive software," *IEEE Softw.*, vol. 34, no. 3, pp. 56–63, Feb. 2017.

[15] S. B. Kotsiantis, "Supervised machine learning: A review of classification techniques," *Informatica*, vol. 31, no. 1, pp. 249–268, 2007.

[16] V. R. Basili, G. Caldiera, and H. D. Rombach, "The goal question metric approach," in *Encyclopedia of Software Engineering*, vol. 2. New York, NY, USA: Wiley, 1994, pp. 528–532. [Online]. Available: http://fub-taslim.googlecode.com/svn/trunk/WEMSE/INSTICC_Conference_Latex/gqm.pdf

[17] I. C. Society, P. Bourque, and R. E. Fairley, *Pierre Bourque, and Richard E Fairley. Guide to the Software Engineering Body of Knowledge (SWE-BOK(R)): Version 3.0*, 3rd ed. Los Alamitos, CA, USA: IEEE Computer Society Press, 2014.

[18] I. Banerjee, B. Nguyen, V. Garousi, and A. Memon, "Graphical user interface (GUI) testing: Systematic mapping and repository," *Inf. Softw. Technol.*, vol. 55, no. 10, pp. 1679–1694, 2013, doi: 10.1016/j.infsof.2013.03.004.

[19] K. Petersen, R. Feldt, S. Mujtaba, and M. Mattsson, "Systematic mapping studies in software engineering," in *Proc. 12th Int. Conf. Eval. Assess. Softw. Eng.*, vol. 17, 2008, p. 10. [Online]. Available: http://www.cse.chalmers.se/%7B~%7Dfeldt/publications/petersen_ease08_sysmap_studies_in_se.pdf

[20] L. Chen, M. A. Babar, and H. Zhang, "Towards an evidence-based understanding of electronic data sources," in *Proc. 14th Int. Conf. Eval. Assessment Softw. Eng. (EASE)*, Apr. 2010, pp. 1–4.

[21] R. Sharma, J. Bhatia, and K. K. Biswas, "Machine learning for constituency test of coordinating conjunctions in requirements specifications," in *Proc. 3rd Int. Workshop Realizing Artif. Intell. Synergies Softw. Eng. (RAISE)*, 2014, pp. 25–31.

[22] M. Singh, V. Anu, G. S. Walia, and A. Goswami, "Validating requirements reviews by introducing fault-type level granularity," in *Proc. 11th Innov. Softw. Eng. Conf.*, Feb. 2018, pp. 1–11.

[23] A. Perini, A. Susi, and P. Avesani, "A machine learning approach to software requirements prioritization," *IEEE Trans. Softw. Eng.*, vol. 39, no. 4, pp. 445–461, Apr. 2013. [Online]. Available: http://ieeexplore.ieee.org/document/6249686/

[24] R. Navarro-Almanza, R. Juarez-Ramirez, and G. Licea, "Towards supporting software engineering using deep learning: A case of software requirements classification," in *Proc. 5th Int. Conf. Softw. Eng. Res. Innov. (CONISOFT)*, Oct. 2017, pp. 116–120.

[25] J. Guo, J. Cheng, and J. Cleland-Huang, "Semantically enhanced software traceability using deep learning techniques," in *Proc. IEEE/ACM 39th Int. Conf. Softw. Eng. (ICSE)*, May 2017, pp. 3–14.

[26] M. White, C. Vendome, M. Linares-Vasquez, and D. Poshyvanyk, "Toward deep learning software repositories," in *Proc. IEEE/ACM 12th Work. Conf. Mining Softw. Repositories*, May 2015, pp. 334–345.

[27] T. T. Nguyen, P. M. Vu, H. V. Pham, and T. T. Nguyen, "Deep learning UI design patterns of mobile apps," in *Proc. 40th Int. Conf. Softw. Eng., New Ideas Emerg. Results*, May 2018, pp. 65–68.

[28] V.-S. Ionescu, "An approach to software development effort estimation using machine learning," in *Proc. 13th IEEE Int. Conf. Intell. Comput. Commun. Process. (ICCP)*, Sep. 2017, pp. 197–203.

[29] M. Alahmadi, J. Hassel, B. Parajuli, S. Haiduc, and P. Kumar, "Accurately predicting the location of code fragments in programming video tutorials using deep learning," in *Proc. 14th Int. Conf. Predictive Models Data Anal. Softw. Eng.*, Oct. 2018, pp. 2–11.

[30] F. Pecorelli, D. Di Nucci, C. De Roover, and A. De Lucia, "On the role of data balancing for machine learning-based code smell detection," in *Proc. 3rd ACM SIGSOFT Int. Workshop Mach. Learn. Techn. Softw. Qual. Eval. (MaLTeSQuE)*, 2019, pp. 19–24.

[31] N. Maneerat and P. Muenchaisri, "Bad-smell prediction from software design model using machine learning techniques," in *Proc. 8th Int. Joint Conf. Comput. Sci. Softw. Eng. (JCSSE)*, May 2011, pp. 331–336.

[32] H. Lal and G. Pahwa, "Code review analysis of software system using machine learning techniques," in *Proc. 11th Int. Conf. Intell. Syst. Control (ISCO)*, Jan. 2017, pp. 8–13.

[33] Y. Kim, J. Park, S.-J. Cho, Y. Nah, S. Han, and M. Park, "Machine learning-based software classification scheme for efficient program similarity analysis," in *Proc. Conf. Res. Adapt. Convergent Syst.*, Oct. 2015, pp. 114–118.

[34] J. Abbineni and O. Thalluri, "Software defect detection using machine learning techniques," in *Proc. 2nd Int. Conf. Trends Electron. Informat. (ICOEI)*, May 2018, pp. 471–475.

[35] Y. A. Alshehri, K. Goseva-Popstojanova, D. G. Dzielski, and T. Devine, "Applying machine learning to predict software fault proneness using change metrics, static code metrics, and a combination of them," in *Proc. SoutheastCon*, Apr. 2018, pp. 1–7.

[36] S. Halali, M. Staron, M. Ochodek, and W. Meding, "Improving defect localization by classifying the affected asset using machine learning," in *Proc. Int. Conf. Softw. Qual.*, vol. 338. Cham, Switzerland: Springer, 2019, pp. 125–148, doi: 10.1007/978-3-030-05767-1.

[37] G. P. Bhandari and R. Gupta, "Machine learning based software fault prediction utilizing source code metrics," in *Proc. IEEE 3rd Int. Conf. Comput., Commun. Secur. (ICCCS)*, Oct. 2018, pp. 40–45.

[38] G. P. Bhandari and R. Gupta, "Measuring the fault predictability of software using deep learning techniques with software metrics," in *Proc. 5th IEEE Uttar Pradesh Sect. Int. Conf. Electr., Electron. Comput. Eng. (UPCON)*, Nov. 2018, pp. 1–6.

[39] R. Bharathi and R. Selvarani, "A machine learning approach for quantifying the design error propagation in safety critical software system," *IETE J. Res.*, vol. ahead-of-print, pp. 1–15, May 2019, doi: 03772063.2019.1611490.

[40] R. Braga, P. S. Neto, R. Rabêlo, J. Santiago, and M. Souza, "A machine learning approach to generate test oracles," in *Proc. 32th Brazilian Symp. Softw. Eng. (SBES)*, 2018, pp. 142–151.

[41] A. Rosenfeld, O. Kardashov, and O. Zang, "Automation of Android applications testing using machine learning activities classification," in *Proc. 5th Int. Conf. Mobile Softw. Eng. Syst. Autom.*, 2017, pp. 122–132. [Online]. Available: http://arxiv.org/abs/1709.00928

[42] G. Huang, Y. Li, Q. Wang, J. Ren, Y. Cheng, and X. Zhao, "Automatic classification method for software vulnerability based on deep neural network," *IEEE Access*, vol. 7, pp. 28291–28298, 2019.

[43] C. Fu, D. Qian, and Z. Luan, "Estimating software energy consumption with machine learning approach by software performance feature," in *Proc. IEEE Int. Conf. Internet Things (iThings) IEEE Green Comput. Commun. (GreenCom) IEEE Cyber, Phys. Social Comput. (CPSCom) IEEE Smart Data (SmartData)*, Jul. 2018, pp. 490–496.

[44] Y. Tamura, M. Matsumoto, and S. Yamada, "Software reliability model selection based on deep learning," in *Proc. Int. Conf. Ind. Eng., Manage. Sci. Appl. (ICIMSA)*, May 2016, pp. 1–5.

[45] R. Kumar, L. E. H. Son, M. Abdel-Basset, I. Priyadarshini, R. Sharma, and H. V. Long, "Deep learning approach for software maintainability metrics prediction," *IEEE Access*, vol. 7, pp. 61840–61855, 2019.

[46] A. Andrzejak and L. Silva, "Using machine learning for non-intrusive modeling and prediction of software aging," in *Proc. NOMS IEEE Netw. Oper. Manage. Symp.*, Apr. 2008, pp. 25–32. [Online]. Available: http://ieeexplore.ieee.org/document/4575113/

[47] D. Chandra and M. Choudhary, "Prophecy of software maintenance effort with univariate and multivariate approach," in *Proc. Int. Conf. Comput., Commun. Autom.*, no. 1, 2017, pp. 876–880.

[48] T. Xie, "The synergy of human and artificial intelligence in software engineering," in *Proc. 2nd Int. Workshop Realizing Artif. Intell. Synergies Softw. Eng. (RAISE)*, May 2013, pp. 4–6.

[49] N. Nascimento, P. Alencar, C. Lucena, and D. Cowan, "Toward human-in-the-loop collaboration between software engineers and machine learning algorithms," in *Proc. IEEE Int. Conf. Big Data (Big Data)*, Dec. 2018, pp. 3534–3540.

[50] R. Pandita, X. Xiao, H. Zhong, T. Xie, S. Oney, and A. Paradkar, "Inferring method specifications from natural language API descriptions," in *Proc. 34th Int. Conf. Softw. Eng. (ICSE)*, Jun. 2012, pp. 815–825.

[51] M. Jahan, Z. S. H. Abad, and B. Far, "Detecting emergent behaviors and implied scenarios in scenario-based specifications: A machine learning approach," in *Proc. IEEE/ACM 11th Int. Workshop Modeling Softw. Eng. (MiSE)*, May 2019, pp. 8–14.

[52] A. Ferrari and A. Esuli, "An NLP approach for cross-domain ambiguity detection in requirements engineering," *Automated Softw. Eng.*, vol. 26, no. 3, pp. 559–598, Sep. 2019, doi: 10.1007/s10515-019-00261-7.

[53] A. K. Dwivedi, A. Tirkey, R. B. Ray, and S. K. Rath, "Software design pattern recognition using machine learning techniques," in *Proc. IEEE Region Conf. (TENCON)*, Nov. 2016, pp. 222–227.

[54] D. Zhang, "Machine learning in value-based software test data generation," in *Proc. 18th IEEE Int. Conf. Tools With Artif. Intell. (ICTAI)*, Nov. 2006, pp. 732–736.

[55] O. Malgonde and K. Chari, "An ensemblebased model for predicting agile software development effort," *Empirical Softw. Eng.*, vol. 24, no. 2, 2019.

[56] L. E. L. B., A. Raj, J. Bosch, and H. Holmstr, "A taxonomy of software engineering challenges for machine learning systems: An empirical investigation," in *Proc. Int. Conf. Agile Softw. Develop.*, vol. 149, May 2013, pp. 227–243, doi: 10.1007/978-3-642-38314-4.

[57] T. Xie, "Dependable software engineering. Theories, tools, and applications," in *Proc. Int. Symp. Dependable Softw. Eng., Theories, Tools, Appl.*, vol. 10998. Beijing, China: Springer, 2018, pp. 3–7, doi: 10.1007/978-3-319-99933-3.

[58] B. Bruegge, J. David, J. Helming, and M. Koegel, "Classification of tasks using machine learning," in *Proc. 5th Int. Conf. Predictor Models Softw. Eng. (PROMISE)*, 2009, pp. 1–11.

[59] W. T. Tsai, K. G. Heisler, D. Volovik, and I. A. Zualkernan, "A critical look at the relationship between AI and software engineering," in *Proc. IEEE Workshop Lang. Automat.@m_Symbiotic Intell. Robot.*, Aug. 1988, pp. 2–18.

[60] H. K. Dam, "Empowering software engineering with artificial intelligence," in *Proc. Austral. Symp. Service Res. Innov. Cham, Switzerland: Springer, 2019, pp. 3–12, doi: 10.1007/978-3-030-32242-7_1.

[61] D. Kalles, "Artificial intelligence meets software engineering in computing education," in *Proc. 9th Hellenic Conf. Artif. Intell.*, May 2016, pp. 1–5.

[62] B. Lin, F. Zampetti, G. Bavota, M. Di Penta, M. Lanza, and R. Oliveto, "Sentiment analysis for software engineering: How far can we go?" in *Proc. 40th Int. Conf. Softw. Eng.*, May 2018, pp. 94–104.

[63] K. Meinke and A. Bennaceur, "Machine learning for software engineering: Models, methods, and applications," in *Proc. 40th Int. Conf. Softw. Eng., Companion*, May 2018, pp. 548–549, doi: 10.1145/3183440.3183461.

[64] D. Michie, "Methodologies from machine learning in data analysis and software," *Comput. J.*, vol. 34, no. 6, pp. 559–565, Dec. 1991.

[65] F. Park, "Artificial intelligence and software engineering: Breaking the toy mold," *Automated Softw. Eng.*, vol. 270, pp. 255–270, Jul. 1997.

[66] R. Robbes and A. Janes, "Leveraging small software engineering data sets with pre-trained neural networks," in *Proc. IEEE/ACM 41st Int. Conf. Softw. Eng., New Ideas Emerg. Results (ICSE-NIER)*, May 2019, pp. 29–32.

[67] B. Schreck, S. Mallapur, S. Damle, N. J. James, S. Vohra, R. Prasad, and K. Veeramachaneni, "Augmenting software project managers with predictions from machine learning," in *Proc. IEEE Int. Conf. Big Data (Big Data)*, Dec. 2018, pp. 2004–2011.

[68] J. Shen, O. Baysal, and M. O. Shafiq, "Evaluating the performance of machine learning sentiment analysis algorithms in software engineering," in *Proc. IEEE Int. Conf. Dependable, Autonomic Secure Comput., Int. Conf. Pervas. Intell. Comput., Int. Conf. Cloud Big Data Comput., Int. Conf. Cyber Sci. Technol. Congr. (DASC/PiCom/CBDCom/CyberSciTech)*, Aug. 2019, pp. 1023–1030.

[69] M. White, "Deep representations for software engineering," in *Proc. IEEE/ACM 37th IEEE Int. Conf. Softw. Eng.*, May 2015, pp. 781–783.

[70] S. N. Ahsan and F. Wotawa, "Impact analysis of SCRs using single and multi-label machine learning classification," in *Proc. ACM-IEEE Int. Symp. Empirical Softw. Eng. Meas. (ESEM)*, Sep. 2010, pp. 1–4.

[71] E.-V. Chioasca, "Using machine learning to enhance automated requirements model transformation," in *Proc. 34th Int. Conf. Softw. Eng. (ICSE)*, Jun. 2012, pp. 1487–1490.

[72] H. Khosrowjerdi and K. Meinke, "Learning-based testing for autonomous systems using spatial and temporal requirements," in *Proc. 1st Int. Workshop Mach. Learn. Softw. Eng. Symbiosis*, Sep. 2018, pp. 6–15.

[73] A. J. Ramirez and B. H. Cheng, "Automatic derivation of utility functions for monitoring software requirements," in *Model Driven Engineering Languages and Systems* (Lecture Notes in Computer Science), vol. 6981. Berlin, Germany: Springer, 2011, pp. 501–516.

[74] F. G. M. Galster and F. Gilson, "What quality attributes can we find in product backlogs? A machine learning perspective," in *Proc. Eur. Conf. Softw. Archit.*, 2019, pp. 88–96.

[75] C.-R. Turliuc, "ProbPoly: A probabilistic inductive logic programming framework with application in model checking," in *Proc. Int. Workshop Mach. Learn. Technol. Softw. Eng. (MALETS)*, 2011, pp. 43–50, doi: 10.1145/2070821.2070827.

[76] S. N. Ahsan, J. Ferzund, and F. Wotawa, "Automatic classification of software change request using multi-label machine learning methods," in *Proc. 33rd Annu. IEEE Softw. Eng. Workshop*, Oct. 2009, pp. 79–86.

[77] E. Dias Canedo and B. Cordeiro Mendes, "Software requirements classification using machine learning algorithms," *Entropy*, vol. 22, no. 9, p. 1057, Sep. 2020.

[78] A. Kobilica, M. Ayub, and J. Hassine, "Automated identification of security requirements: A machine learning approach," in *Proc. ACM Int. Conf. Ser.*, no. 1, 2020, pp. 475–480.

[79] A. Sharma and G. Moranna, "Natural engineering: Applying a genetic computing model to engineering self-aware software," in *Proc. 1st Int. Workshop Softw. Eng. Cognit. Services*, May 2018, pp. 25–28.

[80] M. A. Rahman, M. A. Haque, M. N. A. Tawhid, and M. S. Siddik, "Classifying non-functional requirements using RNN variants for quality software development," in *Proc. 3rd ACM SIGSOFT Int. Workshop Mach. Learn. Techn. Softw. Qual. Eval. (MaLTeSQuE)*, 2019, pp. 25–30.

[81] L. van Rooijen, F. S. Baumer, M. C. Platenius, M. Geierhos, H. Hamann, and G. Engels, "From user demand to software service: Using machine learning to automate the requirements specification process," in *Proc. IEEE 25th Int. Requirements Eng. Conf. Workshops (REW)*, Sep. 2017, pp. 379–385.

[82] J. Cleland-Huang, A. Czauderna, M. Gibiec, and J. Emenecker, "A machine learning approach for tracing regulatory codes to product specific requirements," in *Proc. 32nd ACM/IEEE Int. Conf. Softw. Eng. (ICSE)*, May 2010, p. 155.

[83] A. C. Marcén, R. Lapeña, Ó. Pastor, and C. Cetina, "Traceability link recovery between requirements and models using an evolutionary algorithm guided by a learning to rank algorithm: Train control and management case," *J. Syst. Softw.*, vol. 163, May 2020, Art. no. 110519.

[84] C. Mills and S. Haiduc, "A machine learning approach for determining the validity of traceability links," in *Proc. IEEE/ACM 39th Int. Conf. Softw. Eng. Companion (ICSE-C)*, May 2017, pp. 121–123.

[85] K. W. Nafi, B. Roy, C. K. Roy, and K. A. Schneider, "CroLSim: Cross language software similarity detector using API documentation," in *Proc. 18th Int. Work. Conf. Source Code Anal. Manipulation (CroLSim)*, Sep. 2018, pp. 139–148.

[86] M. Wieloch, S. Amornborvornwong, and J. Cleland-Huang, "Trace-by-classification: A machine learning approach to generate trace links for frequently occurring software artifacts," in *Proc. 7th Int. Workshop Traceability Emerg. Forms Softw. Eng. (TEFSE)*, May 2013, pp. 110–114.

[87] S. Ahn, J. Kim, and S. Kang, "A novel shared memory framework for distributed deep learning in high-performance computing architecture," in *Proc. 40th Int. Conf. Softw. Eng., Companion*, May 2018, pp. 191–192.

[88] Y. Boubekeur, G. Mussbacher, and S. McIntosh, "Automatic assessment of students' software models using a simple heuristic and machine learning," in *Proc. 23rd ACM/IEEE Int. Conf. Model Driven Eng. Lang. Syst., Companion*, Oct. 2020, pp. 84–93.

[89] O. Castro-Lopez and I. F. Vega-Lopez, "Fast deployment and scoring of support vector machine models in CPU and GPU," in *Proc. 1st Int. Workshop Mach. Learn. Softw. Eng. Symbiosis*, Sep. 2018, pp. 45–52.

[90] S. Chollet, P. Lalanda, and J. Bardin, "Software service recommendation base on collaborative filtering neural network model," in *Proc. Int. Conf. Service-Oriented Comput.* Cham, Switzerland: Springer, 2013, pp. 1–20, doi: 10.1007/978-3-030-03596-9_28.

[91] W. S. Cunha, G. A. Armijo, and V. V. de Camargo, "InSet: A tool to identify architecture smells using machine learning," in *Proc. 34th Brazilian Symp. Softw. Eng.*, Oct. 2020, pp. 760–765.

[92] A. Dawoud, S. Shahristani, and C. Raun, "Deep learning and software-defined networks: Towards secure IoT architecture," *Internet Things*, vols. 3–4, pp. 82–89, Oct. 2018, doi: 10.1016/j.iot.2018.09.003.

[93] Q. Factors, L.-C. Revised, J. Hernández-orallo, and M. J. Ramírez-quintana, "Software as learning: Quality factors and life-cycle revised," in *Proc. Int. Conf. Fundam. Approaches Softw. Eng.*, 2000, pp. 147–162.

[94] P. G. Mousouliotis and P. L. Petrou, *Software-Defined FPGA Accelerator Design for Mobile Deep Learning Applications*, vol. 11444. Darmstadt, Germany: Springer, 2019, doi: 10.1007/978-3-030-17227-5.

[95] H. Muccini and K. Vaidhyanathan, "ArchLearner: Leveraging machine-learning techniques for proactive architectural adaptation," in *Proc. 13th Eur. Conf. Softw. Archit. (ECSA)*, vol. 2, 2019, pp. 38–41, doi: 10.1145/3344948.3344962.

[96] K. O. Neal, P. Brisk, A. Abousamra, Z. Waters, E. Shriver, and I. Corporation, "GPU performance estimation using software rasterization," *Trans. Embedded Comput. Syst.*, vol. 16, no. 5, pp. 1–21, 2017.

[97] R. Clarisó and J. Cabot, "Applying graph kernels to model-driven engineering problems," in *Proc. 1st Int. Workshop Mach. Learn. Softw. Eng. Symbiosis*, Sep. 2018, pp. 1–5.

[98] A. Sankaran, R. Aralikatte, S. Mani, S. Khare, N. Panwar, and N. Gantayat, "DARVIZ: Deep abstract representation, visualization, and verification of deep learning models," in *Proc. IEEE/ACM 39th Int. Conf. Softw. Eng., New Ideas Emerg. Technol. Results Track (ICSE-NIER)*, May 2017, pp. 47–50.

[99] M. Shin and A. L. Goel, "Modeling software component criticality using a machine learning approach," in *Proc. Int. Conf. AI, Simulation, Planning High Autonomy Syst.*, 2005, pp. 440–448.

[100] B. K. Sidhu, K. Singh, and N. Sharma, "A catalogue of model smells and refactoring operations for object-oriented software," in *Proc. 2nd Int. Conf. Inventive Commun. Comput. Technol. (ICICCT)*, Apr. 2018, pp. 313–319.

[101] Q. Song, X. Zhu, G. Wang, H. Sun, H. Jiang, C. Xue, B. Xu, and W. Song, "A machine learning based software process model recommendation method," *J. Syst. Softw.*, vol. 118, pp. 85–100, Aug. 2016.

[102] X. WU and Z. ZHOU, "Model reuse with domain knowledge," in *Proc. 12th ACM/IEEE Int. Symp. Empirical Softw. Eng. Meas.*, vol. 47, no. 11, Nov. 2017, pp. 1483–1492.

[103] C. L. Prabha and N. Shivakumar, "Improving design quality of software using machine learning techniques," in *Proc. 6th Int. Conf. Adv. Comput. Commun. Syst. (ICACCS)*, Mar. 2020, pp. 583–588.

[104] R. Rajaraman, P. K. Kapur, and D. Kumar, "Determining software inter-dependency patterns for integration testing by applying machine learning on logs and telemetry data," in *Proc. 8th Int. Conf. Rel., INFO-COM Technol. Optim. (Trends Future Directions) (ICRITO)*, Jun. 2020, pp. 1080–1084.

[105] H. Thaller, L. Linsbauer, and A. Egyed, "Feature maps: A comprehensible software representation for design pattern detection," in *Proc. IEEE 26th Int. Conf. Softw. Anal., Evol. Reeng. (SANER)*, Feb. 2019, pp. 207–217.

[106] N. S. V. Borozanov and S. Hacks, "Using machine learning techniques for evaluating the similarity of enterprise architecture models," in *Proc. Int. Conf. Adv. Inf. Syst. Eng.* Cham, Switzerland: Springer, 2019, pp. 563–578.

[107] M. M. Al Asheeri and M. Hammad, "Machine learning models for software cost estimation," in *Proc. Int. Conf. Innov. Intell. Informat., Comput., Technol. (3ICT)*, Sep. 2019, pp. 1–6.

[108] B. Baskeles, B. Turhan, and A. Bener, "Software effort estimation using machine learning methods," in *Proc. 22nd Int. Symp. Comput. Inf. Sci.*, Nov. 2007, pp. 1–6.

[109] A. BaniMustafa, "Predicting software effort estimation using machine learning techniques," in *Proc. 8th Int. Conf. Comput. Sci. Inf. Technol. (CSIT)*, Jul. 2018, pp. 249–256.

[110] A. Corazza, S. Di Martino, F. Ferrucci, C. Gravino, and E. Mendes, "Using support vector regression for web development effort estimation," in *Software Process and Product Measurement* (Lecture Notes in Computer Science), vol. 5891. Berlin, Germany: Springer, 2009, pp. 255–271.

[111] E. Ertuğrul, Z. Baytar, Ç. Çatal, and Ö. C. Muratli, "Performance tuning for machine learning-based software development effort prediction models," *TURKISH J. Electr. Eng. Comput. Sci.*, vol. 27, pp. 1308–1324, Mar. 2019. [Online]. Available: http://online.journals.tubitak.gov.tr/openDoiPdf.htm?mKodu=elk-1809-129

[112] J. Huang, Y.-F. Li, and M. Xie, "An empirical analysis of data preprocessing for machine learning-based software cost estimation," *Inf. Softw. Technol.*, vol. 67, pp. 108–127, Nov. 2015, doi: 10.1016/j.infsof.2015.07.004.

[113] A. Idri, A. Zahi, E. Mendes, and A. Zakrani, "Software cost estimation models using radial basis function neural networks," in *Software Process and Product Measurement* (Lecture Notes in Computer Science), vol. 4895. Berlin, Germany: Springer, 2008, pp. 21–31.

[114] K. Iwata, T. Nakashima, Y. Anan, and N. Ishii, "Effort estimation for embedded software development projects by combining machine learning with classification," in *Proc. 4th Int. Conf. Appl. Comput. Inf. Technol./3rd Int. Conf. Comput. Sci./Intell. Appl. Inform./1st Int. Conf. Big Data, Cloud Comput., Data Sci. Eng. (ACIT-CSII-BCD)*, Dec. 2016, pp. 265–270.

[115] O. Mizuno, T. Hamasaki, Y. Takagi, and T. Kikuno, "An empirical evaluation of predicting runaway software projects using Bayesian classification," in *Proc. Int. Conf. Product Focused Softw. Process Improvement*, 2010, pp. 263–273.

[116] J. Murillo-Morera, C. Quesada-López, C. Castro-Herrera, and M. Jenkins, "A genetic algorithm based framework for software effort prediction," *J. Softw. Eng. Res. Develop.*, vol. 5, no. 1, pp. 1–33, Dec. 2017.

[117] S. M. Satapathy and S. K. Rath, "Empirical assessment of machine learning models for effort estimation of web-based applications," in *Proc. 10th Innov. Softw. Eng. Conf.*, Feb. 2017, pp. 74–84.

[118] P. Sharma and J. Singh, "Systematic literature review on software effort estimation using machine learning approaches," in *Proc. Int. Conf. Next Gener. Comput. Inf. Syst. (ICNGCIS)*, Dec. 2017, pp. 43–47.

[119] E. N. Regolin, G. A. de Souza, A. R. T. Pozo, and S. R. Vergilio, "Exploring machine learning techniques for software size estimation," in *Proc. 23rd Int. Conf. Chilean Comput. Sci. Soc. (SCCC)*, Nov. 2003, pp. 130–136.

[120] K. Srinivasan and D. Fisher, "Machine learning approaches to estimating software development effort," *IEEE Trans. Softw. Eng.*, vol. 21, no. 2, pp. 126–137, Feb. 1995.

[121] J. Shivhare and S. K. Rath, "Software effort estimation using machine learning techniques," in *Proc. 7th India Softw. Eng. Conf. (ISEC)*, Jan. 2014, pp. 92–98.

[122] J. Wen, S. Li, Z. Lin, Y. Hu, and C. Huang, "Systematic literature review of machine learning based software development effort estimation models," *Inf. Softw. Technol.*, vol. 54, no. 1, pp. 41–59, 2012, doi: 10.1016/j.infsof.2011.09.002. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0950584911001832

[123] I. Wright and A. Ziegler, "The standard coder: A machine learning approach to measuring the effort required to produce source code change," in *Proc. IEEE/ACM 7th Int. Workshop Realizing Artif. Intell. Synergies Softw. Eng. (RAISE)*, May 2019, pp. 1–7.

[124] J. Cambronero, H. Li, S. Kim, K. Sen, and S. Chandra, "When deep learning met code search," in *Proc. 27th ACM Joint Meeting Eur. Softw. Eng. Conf. Symp. Found. Softw. Eng.*, Aug. 2019, pp. 964–974.

[125] B. Gelman, B. Hoyle, J. Moore, J. Saxe, and D. Slater, "A language-agnostic model for semantic source code labeling," in *Proc. 1st Int. Workshop Mach. Learn. Softw. Eng. Symbiosis*, Sep. 2018, pp. 36–44.

[126] G. Mostaeen, J. Svajlenko, B. Roy, C. K. Roy, and K. A. Schneider, "CloneCognition: Machine learning based code clone validation tool," in *Proc. 27th ACM Joint Meeting Eur. Softw. Eng. Conf. Symp. Found. Softw. Eng.*, Aug. 2019, pp. 1105–1109.

[127] J. Ott, A. Atchison, P. Harnack, A. Bergh, and E. Linstead, "A deep learning approach to identifying source code in images and video," in *Proc. 15th Int. Conf. Mining Softw. Repositories*, May 2018, pp. 376–386.

[128] T. Parr and J. Vinju, "Towards a universal code formatter through machine learning," in *Proc. ACM SIGPLAN Int. Conf. Softw. Lang. Eng.*, Oct. 2016, pp. 137–151.

[129] M. E. H. Schumacher, K. T. Le, and A. Andrzejak, "Improving code recommendations by combining neural and classical machine learning approaches," in *Proc. IEEE/ACM 42nd Int. Conf. Softw. Eng. Workshops*, Jun. 2020, pp. 476–482.

[130] A. M. Sheneamer, "An automatic advisor for refactoring software clones based on machine learning," *IEEE Access*, vol. 8, pp. 124978–124988, 2020.

[131] M. White, M. Tufano, C. Vendome, and D. Poshyvanyk, "Deep learning code fragments for code clone detection," in *Proc. 31st IEEE/ACM Int. Conf. Automated Softw. Eng.*, Aug. 2016, pp. 87–98.

[132] J. Yang, K. Hotta, Y. Higo, H. Igaki, and S. Kusumoto, "Filtering clones for individual user based on machine learning analysis," in *Proc. 6th Int. Workshop Softw. Clones (IWSC)*, Jun. 2012, pp. 76–77.

[133] U. Azadi, F. A. Fontana, and M. Zanoni, "Machine learning based code smell detection through WekaNose," in *Proc. 40th Int. Conf. Softw. Eng., Companion*, May 2018, pp. 29–32.

[134] M. Y. Mhawish and M. Gupta, "Predicting code smells and analysis of predictions: Using machine learning techniques and software metrics," *J. Comput. Sci. Technol.*, vol. 35, no. 6, pp. 1428–1445, Nov. 2020.

[135] N. Ayesha and N. G. Yethiraj, "Review on code examination proficient system in software engineering by using machine learning approach," in *Proc. Int. Conf. Inventive Res. Comput. Appl. (ICIRCA)*, Jul. 2018, pp. 324–327.

[136] K. T. Ayinala, K. S. Cheng, K. Oh, T. Song, and M. Song, "Code inspection support for recurring changes with deep learning in evolving software," in *Proc. IEEE 44th Annu. Comput., Softw., Appl. Conf. (COMPSAC)*, Jul. 2020, pp. 931–942.

[137] K. Chandra, G. Kapoor, R. Kohli, and A. Gupta, "Improving software quality using machine learning," in *Proc. Int. Conf. Innov. Challenges Cyber Secur. (ICICCS-INBUSH)*, Feb. 2016, pp. 115–118.

[138] G. Fouqu and C. Vrain, "Building a tool for software code analysis a machine learning approach structures," in *Proc. Int. Conf. Adv. Inf. Syst. Eng.*, 1992, pp. 278–289.

[139] V. Geist, M. Moser, J. Pichler, S. Beyer, and M. Pinzger, "Leveraging machine learning for software redocumentation," in *Proc. IEEE 27th Int. Conf. Softw. Anal., Evol. Reeng. (SANER)*, Feb. 2020, pp. 622–626.

[140] M. Madera and R. Tomoń, "A case study on machine learning model for code review expert system in software engineering," in *Proc. Federated Conf. Comput. Sci. Inf. Syst.*, Sep. 2017, pp. 1357–1363.

[141] M. Naseer, W. Zhang, and W. Zhu, "Prediction of coding intricacy in a software engineering team through machine learning to ensure cooperative learning and sustainable education," *Sustainability*, vol. 12, no. 21, pp. 1–15, 2020.

[142] A. Trivedi, J. S. Thakur, and A. Gupta, "Code nano-pattern detection using deep learning," in *Proc. 13th Innov. Softw. Eng. Conf. Formerly Known India Softw. Eng. Conf.*, Feb. 2020, pp. 1–6.

[143] A. LeClair, Z. Eberhart, and C. Mcmillan, "Adapting neural text classification for improved software categorization," in *Proc. IEEE Int. Conf. Softw. Maintenance Evol. (ICSME)*, Sep. 2018, pp. 461–472.

[144] Z. Sun, Y. Liu, Z. Cheng, C. Yang, and P. Che, "Req2Lib: A semantic neural model for software library recommendation," 2020, *arXiv:2005.11757*. [Online]. Available: http://arxiv.org/abs/2005.11757

[145] M. Tufano, C. Watson, G. Bavota, M. Di Penta, M. White, and D. Poshyvanyk, "Deep learning similarities from different representations of source code," in *Proc. 15th Int. Conf. Mining Softw. Repositories*, May 2018, pp. 542–553.

[146] X. Zhang and K. Ben, "A neural language model with a modified attention," in *Proc. 9th Int. Conf. Softw. Eng. Service Sci.*, Nov. 2018, pp. 232–236.

[147] G. Zhao and J. Huang, "DeepSim: Deep learning code functional similarity," in *Proc. 26th ACM Joint Meeting Eur. Softw. Eng. Conf. Symp. Found. Softw. Eng.*, Oct. 2018, pp. 141–151.

[148] S. M. Abozeed, M. Y. ElNainay, S. A. Fouad, and M. S. Abougabal, "Software bug prediction employing feature selection and deep learning," in *Proc. Int. Conf. Adv. Emerg. Comput. Technol. (AECT)*, Feb. 2020, pp. 1–6.

[149] M. Borg, O. Svensson, K. Berg, and D. Hansson, "SZZ unleashed: An open implementation of the SZZ algorithm–featuring example usage in a study of just-in-time bug prediction for the Jenkins project," in *Proc. 3rd ACM SIGSOFT Int. Workshop Mach. Learn. Techn. Softw. Qual. Eval. (MaLTeSQuE)*, 2019, pp. 7–12.

[150] Y. Brun and M. D. Ernst, "Finding latent code errors via machine learning over program executions," in *Proc. 26th Int. Conf. Softw. Eng.*, May 2004, pp. 480–490.

[151] M. Cerrada, F. Pacheco, R. V. Sanchez, D. Cabrera, J. Macancela, and P. Lucero, "SOA based integrated software to develop fault diagnosis models using machine learning in rotating machinery," in *Proc. IEEE Symp. Service-Oriented Syst. Eng. (SOSE)*, Apr. 2017, pp. 28–37.

[152] E. Ceylan, F. O. Kutlubay, and A. B. Bener, "Software defect identification using machine learning techniques," in *Proc. 32nd EUROMI-CRO Conf. Softw. Eng. Adv. Appl. (EUROMICRO)*, Aug. 2006, pp. 240–247.

[153] V. U. B. Challagulla, F. B. Bastani, I.-L. Yen, and R. A. Paul, "Empirical assessment of machine learning based software defect prediction techniques," in *Proc. 10th IEEE Int. Workshop Object-Oriented Real-Time Dependable Syst.*, 2005, pp. 389–400.

[154] J. Chen, K. Hu, Y. Yu, Z. Chen, Q. Xuan, Y. Liu, and V. Filkov, "Software visualization and deep transfer learning for effective software defect prediction," in *Proc. ACM/IEEE 42nd Int. Conf. Softw. Eng.*, Jun. 2020, pp. 578–589.

[155] C. J. Clemente, F. Jaafar, and Y. Malik, "Is predicting software security bugs using deep learning better than the traditional machine learning algorithms?" in *Proc. IEEE Int. Conf. Softw. Qual., Rel. Secur. (QRS)*, Jul. 2018, pp. 95–102.

[156] H. K. Dam, T. Pham, S. W. Ng, T. Tran, J. Grundy, A. Ghose, T. Kim, and C.-J. Kim, "Lessons learned from using a deep tree-based model for software defect prediction in practice," in *Proc. IEEE/ACM 16th Int. Conf. Mining Softw. Repositories (MSR)*, May 2019, pp. 46–57.

[157] A. Dwarakanath, M. Ahuja, S. Sikand, R. M. Rao, R. P. J. C. Bose, N. Dubash, and S. Podder, "Identifying implementation bugs in machine learning based image classifiers using metamorphic testing," in *Proc. 27th ACM SIGSOFT Int. Symp. Softw. Test. Anal.*, Jul. 2018, pp. 118–128, doi: 10.1145/3213846.3213858.

[158] J. Ferzund, S. N. Ahsan, and F. Wotawa, "Analysing bug prediction capabilities of static code metrics in open source software," in *Software Process and Product Measurement* (Lecture Notes in Computer Science), vol. 5338. Berlin, Germany: Springer, 2008, pp. 331–343.

[159] A. Gupta, S. Sharma, S. Goyal, and M. Rashid, "Novel XGBoost tuned machine learning model for software bug prediction," in *Proc. Int. Conf. Intell. Eng. Manage. (ICIEM)*, Jun. 2020, pp. 376–380.

[160] T. Hall and D. Bowes, "The state of machine learning methodology in software fault prediction," in *Proc. 11th Int. Conf. Mach. Learn. Appl.*, Dec. 2012, pp. 308–313.

[161] K. Heo, H. Oh, and K. Yi, "Machine-learning-guided selectively unsound static analysis," in *Proc. IEEE/ACM 39th Int. Conf. Softw. Eng. (ICSE)*, May 2017, pp. 519–529.

[162] X. Huo, Y. Yang, M. Li, and D.-C. Zhan, "Learning semantic features for software defect prediction by code comments embedding," in *Proc. IEEE Int. Conf. Data Mining (ICDM)*, Nov. 2018, pp. 1049–1054.

[163] V. Jindal, "Towards an intelligent fault prediction code editor to improve software quality using deep learning," in *Proc. Conf. Companion 2nd Int. Conf. Art, Sci., Eng. Program.*, Apr. 2018, pp. 222–223.

[164] J. Kahles, J. Torronen, T. Huuhtanen, and A. Jung, "Automating root cause analysis via machine learning in agile software testing environments," in *Proc. 12th IEEE Conf. Softw. Test., Validation Verification (ICST)*, Apr. 2019, pp. 379–390.

[165] N. M. Kalibhat, S. Varshini, C. Kollengode, D. Sitaram, and S. Kalambur, "Software troubleshooting using machine learning," in *Proc. IEEE 24th Int. Conf. High Perform. Comput. Workshops (HiPCW)*, Dec. 2017, pp. 3–10.

[166] R. P and P. Kambli, "Predicting bug in a software using ANN based machine learning techniques," in *Proc. IEEE Int. Conf. Innov. Technol. (INOCON)*, Nov. 2020, pp. 1–5.

[167] S. Karim, H. Leslie, H. Spits, E. Abdurachman, and B. Soewito, "Software metrics for fault prediction using machine learning approaches," in *Proc. IEEE Int. Conf. Cybern. Comput. Intell.*, Nov. 2017, pp. 19–23.

[168] M. Kim, J. Nam, J. Yeon, S. Choi, and S. Kim, "REMI: Defect prediction for efficient API testing," in *Proc. 10th Joint Meeting Found. Softw. Eng.*, Aug. 2015, pp. 990–993.

[169] A. Kumar and A. Bansal, "Software fault proneness prediction using genetic based machine learning techniques," in *Proc. 4th Int. Conf. Internet Things, Smart Innov. Usages (IoT-SIU)*, Apr. 2019, pp. 1–5.

[170] H. Lal and G. Pahwa, "Root cause analysis of software bugs using machine learning techniques," in *Proc. 7th Int. Conf. Cloud Comput., Data Sci. Eng. (Confluence)*, Jan. 2017, pp. 105–111.

[171] J. Li, P. He, J. Zhu, and M. R. Lyu, "Software defect prediction via convolutional neural network," in *Proc. IEEE Int. Conf. Softw. Qual., Rel. Secur. (QRS)*, Jul. 2017, pp. 318–328.

[172] H. Liang, Y. Yu, L. Jiang, and Z. Xie, "Seml: A semantic LSTM model for software defect prediction," *IEEE Access*, vol. 7, pp. 83812–83824, 2019.

[173] Y. Luo, K. Ben, and L. Mi, "Software metrics reduction for fault-proneness prediction of software modules," in *Network and Parallel Computing* (Lecture Notes in Computer Science), vol. 6289. Berlin, Germany: Springer, 2010, pp. 432–441.

[174] R. Malhotra, L. Bahl, S. Sehgal, and P. Priya, "Empirical comparison of machine learning algorithms for bug prediction in open source software," in *Proc. Int. Conf. Big Data Anal. Comput. Intell. (ICBDAC)*, Mar. 2017, pp. 40–45.

[175] A. Majd, M. Vahidi-Asl, A. Khalilian, P. Poorsarvi-Tehrani, and H. Haghighi, "SLDeep: Statement-level software defect prediction using deep-learning model on static code features," *Expert Syst. Appl.*, vol. 147, Jun. 2020, Art. no. 113156, doi: 10.1016/j.eswa.2019.113156.

[176] C. Ni, W.-S. Liu, X. Chen, Q. Gu, D.-X. Chen, and Q.-G. Huang, "A cluster based feature selection method for cross-project software defect prediction," *J. Comput. Sci. Technol.*, vol. 32, no. 6, pp. 1090–1107, Nov. 2017.

[177] A. V. Phan and M. Le Nguyen, "Convolutional neural networks on assembly code for predicting software defects," in *Proc. 21st Asia Pacific Symp. Intell. Evol. Syst. (IES)*, Nov. 2017, pp. 37–42.

[178] A. V. Phan, M. Le Nguyen, and L. T. Bui, "Convolutional neural networks over control flow graphs for software defect prediction," in *Proc. IEEE 29th Int. Conf. Tools With Artif. Intell. (ICTAI)*, Nov. 2017, pp. 45–52.

[179] T. M. P. Ha, D. H. Tran, L. T. M. Hanh, and N. T. Binh, "Experimental study on software fault prediction using machine learning model," in *Proc. 11th Int. Conf. Knowl. Syst. Eng. (KSE)*, Oct. 2019, pp. 1–5.

[180] S. Polisetty, A. Miranskyy, and A. Başar, "On usefulness of the deep-learning-based bug localization models to practitioners," in *Proc. 15th Int. Conf. Predictive Models Data Anal. Softw. Eng.*, Sep. 2019, pp. 16–25.

[181] O. A. Qasem, M. Akour, and M. Alenezi, "The influence of deep learning algorithms factors in software fault prediction," *IEEE Access*, vol. 8, pp. 63945–63960, 2020.

[182] P. Deep Singh and A. Chug, "Software defect prediction analysis using machine learning algorithms," in *Proc. 7th Int. Conf. Cloud Comput., Data Sci. Eng. (Confluence)*, Jan. 2017, pp. 775–781.

[183] C. L. Prabha and N. Shivakumar, "Software defect prediction using machine learning techniques," in *Proc. 4th Int. Conf. Trends Electron. Informat. (ICOEI)*, Jun. 2020, pp. 728–733.

[184] R. Rana, M. Staron, J. Hansson, M. Nilsson, and W. Meding, "A framework for adoption of machine learning in industry for software defect prediction," in *Proc. 9th Int. Conf. Softw. Eng. Appl.*, 2014, pp. 383–392.

[185] S. D. Immaculate, M. F. Begam, and M. Floramary, "Software bug prediction using supervised machine learning algorithms," in *Proc. Int. Conf. Data Sci. Commun. (IconDSC)*, Mar. 2019, pp. 1–7.

[186] M. Shepperd, T. Hall, D. Bowes, and T. Hall, "Researcher bias: The use of machine learning in software defect prediction," *Trans. Softw. Eng.*, vol. 40, no. 6, pp. 603–616, 2014.

[187] S. Shukla, R. K. Behera, S. Misra, and S. K. Rath, "Software reliability assessment using deep learning technique," in *Proc. Int. Conf. Comput. Sci. Appl.* Singapore: Springer, 2018, pp. 57–68, doi: 10.1007/978-981-13-2348-5.

[188] Y. Singh, A. Kaur, and R. Malhotra, "Predicting software fault proneness model using neural network," in *Product-Focused Software Process Improvement* (Lecture Notes in Computer Science), vol. 5089, 2008, pp. 204–214.

[189] P. Singh and R. Malhotra, "Assessment of machine learning algorithms for determining defective classes in an object-oriented software," in *Proc. 6th Int. Conf. Rel., Infocom Technol. Optim. Trends Future Directions (ICRITO)*, Sep. 2017, pp. 204–209.

[190] R. Singh, J. Singh, M. S. Gill, R. Malhotra, and Garima, "Transfer learning code vectorizer based machine learning models for software defect prediction," in *Proc. Int. Conf. Comput. Perform. Eval. (ComPE)*, Jul. 2020, pp. 497–502.

[191] U. Subbiah, M. Ramachandran, and Z. Mahmood, "Software engineering framework for software defect management using machine learning techniques with azure," in *Software Engineering in the Era of Cloud Computing*. Cham, Switzerland: Springer, 2020, doi: 10.1007/978-3-030-33624-0_7.

[192] D. Sudharson and D. Prabha, "A novel machine learning approach for software reliability growth modelling with Pareto distribution function," *Soft Comput.*, vol. 23, no. 18, pp. 8379–8387, Sep. 2019, doi: 10.1007/s00500-019-04047-7.

[193] Y. Sun, L. Xu, Y. Li, L. Guo, Z. Ma, and Y. Wang, "Utilizing deep architecture networks of VAE in software fault prediction," in *Proc. IEEE Int. Conf. Parallel Distrib. Process. With Appl., Ubiquitous Comput. Commun., Big Data Cloud Comput., Social Comput. Netw., Sustain. Comput. Commun. (ISPA/IUCC/BDCloud/SocialCom/SustainCom)*, Dec. 2018, pp. 870–877.

[194] K. Tanaka, A. Monden, and Z. Yucel, "Prediction of software defects using automated machine learning," in *Proc. 20th IEEE/ACIS Int. Conf. Softw. Eng., Artif. Intell., Netw. Parallel/Distributed Comput. (SNPD)*, Jul. 2019, pp. 490–494.

[195] J. Tian and Y. Tian, "A model based on program slice and deep learning for software defect prediction," in *Proc. 29th Int. Conf. Comput. Commun. Netw. (ICCCN)*, Aug. 2020, pp. 1–6.

[196] H. Thaller, L. Linsbauer, A. Egyed, and S. Fischer, "Towards fault localization via probabilistic software modeling," 2020, *arXiv:2001.07409*. [Online]. Available: http://arxiv.org/abs/2001.07409

[197] S. Wang, T. Liu, J. Nam, and L. Tan, "Deep semantic feature learning for software defect prediction," *IEEE Trans. Softw. Eng.*, vol. 46, no. 12, pp. 1267–1293, Dec. 2020.

[198] M. Wen, R. Wu, and S.-C. C. Cheung, "How well do change sequences predict defects? Sequence learning from software changes," *IEEE Trans. Softw. Eng.*, vol. 46, no. 11, pp. 1155–1175, Nov. 2020.

[199] K. Wongpheng and P. Visutsak, "Software defect prediction using convolutional neural network," in *Proc. 35th Int. Tech. Conf. Circuits/Syst., Comput. Commun. (ITC-CSCC)*, Jul. 2020, pp. 240–243.

[200] Z. Xu, K. Ren, S. Qin, and F. Craciun, "CDGDroid: Android malware detection based on deep learning using CFG and DFG," in *Proc. Int. Conf. Formal Eng. Methods*, vol. 3308. Cham, Switzerland: Springer, 2004, pp. 177–193. [Online]. Available: http://link.springer.com/10.1007/b102837

[201] S. Yang, S. Yang, Z. Fang, X. Yu, L. Rui, and Y. Ma, "Fault prediction for software system in industrial internet: A deep learning algorithm via effective dimension reduction," in *Proc. Int. Conf. Cyber-Living, Cyber-Syndrome Cyber-Health (CCIS)*, vol. 1137. Singapore: Springer, 2019, pp. 572–580, doi: 10.1007/978-981-15-1922-2_40.

[202] X. Zhang, K. Ben, and J. Zeng, "Cross-entropy: A new metric for software defect prediction," in *Proc. IEEE Int. Conf. Softw. Qual., Rel. Secur. (QRS)*, Jul. 2018, pp. 111–122.

[203] L. Zhao, Z. Shang, L. Zhao, A. Qin, and Y. Y. Tang, "Siamese dense neural network for software defect prediction with small data," *IEEE Access*, vol. 7, pp. 7663–7677, 2018.

[204] F. S. Babamir, A. Hatamizadeh, and S. M. Babamir, "Application of genetic algorithm in automatic software testing," in *Proc. Int. Conf. Netw. Digit. Technol.*, 2010, pp. 545–552.

[205] L. C. Briand, "Novel applications of machine learning in software testing," in *Proc. 8th Int. Conf. Qual. Softw.*, Aug. 2008, pp. 3–10.

[206] L. Ma, F. Juefei-Xu, F. Zhang, J. Sun, M. Xue, B. Li, C. Chen, T. Su, L. Li, Y. Liu, J. Zhao, and Y. Wang, "DeepGauge: Multi-granularity testing criteria for deep learning systems," in *Proc. 33rd ACM/IEEE Int. Conf. Automated Softw. Eng.*, Sep. 2018, pp. 120–131, doi: 10.1145/3238147.3238202.

[207] V. A. de Santiago, L. A. R. da Silva, and P. R. de Andrade Neto, "Testing environmental models supported by machine learning," in *Proc. 3rd Brazilian Symp. Systematic Automated Softw. Test. (SAST)*, 2018, pp. 3–12.

[208] N. Oster, "Automated generation and evaluation of dataflow-based test data for object-oriented software," in *Quality of Software Architectures and Software Quality* (Lecture Notes in Computer Science), vol. 3712. Berlin, Germany: Springer, 2005, pp. 212–226.

[209] C. Paduraru, M.-C. Melemciuc, and M. Paduraru, "Automatic test data generation for a given set of applications using recurrent neural networks," in *Proc. Int. Conf. Softw. Technol.*, vol. 1. Cham, Switzerland: Springer, 2019, pp. 307–326, doi: 10.1007/978-3-030-29157-0_14.

[210] R. Gove and J. Faytong, "Machine learning and event-based software testing: Classifiers for identifying infeasible GUI event sequences," *Adv. Comput.*, vol. 86, pp. 109–135, Jan. 2012, doi: 10.1016/B978-0-12-396535-6.00004-1.

[211] G. Hu, L. Zhu, and J. Yang, "AppFlow: Using machine learning to synthesize robust, reusable UI tests," in *Proc. 26th ACM Joint Meeting Eur. Softw. Eng. Conf. Symp. Found. Softw. Eng.*, Oct. 2018, pp. 269–282.

[212] A. Rafael Lenz, A. Pozo, and S. Regina Vergilio, "Linking software testing results with a machine learning approach," *Eng. Appl. Artif. Intell.*, vol. 26, nos. 5–6, pp. 1631–1640, May 2013, doi: 10.1016/j.engappai.2013.01.008.

[213] W. Zheng, Y. Bai, and H. Che, "A computer-assisted instructional method based on machine learning in software testing class," *Comput. Appl. Eng. Educ.*, vol. 26, no. 5, pp. 1150–1158, Sep. 2018, doi: 10.1002/cae.21962.

[214] J. Alonso, L. Belanche, and D. R. Avresky, "Predicting software anomalies using machine learning techniques," in *Proc. IEEE 10th Int. Symp. Netw. Comput. Appl.*, Aug. 2011, pp. 163–170.

[215] A. Alotaibi, "Identifying malicious software using deep residual long-short term memory," *IEEE Access*, vol. 7, pp. 163128–163137, 2019.

[216] F. Bisio, P. Gastaldo, R. Zunino, and S. Decherchi, "Semi-supervised machine learning approach for unknown malicious software detection," in *Proc. IEEE Int. Symp. Innov. Intell. Syst. Appl. (INISTA)*, Jun. 2014, pp. 52–59.

[217] H. K. Dam, T. Tran, T. Pham, S. W. Ng, J. Grundy, and A. Ghose, "Automatic feature learning for predicting vulnerable software components," *IEEE Trans. Softw. Eng.*, vol. 14, no. 8, pp. 1–19, 2015.

[218] S. K. Dey, M. M. Rahman, and M. R. Uddin, "Detection of flow based anomaly in OpenFlow controller: Machine learning approach in software defined networking," in *Proc. 4th Int. Conf. Electr. Eng. Inf. Commun. Technol. (iCEEiCT)*, Sep. 2018, pp. 416–421.

[219] S. M. Ghaffarian and H. R. Shahriari, "Software vulnerability analysis and discovery using machine-learning and data-mining techniques: A survey," *ACM Comput. Surv.*, vol. 50, no. 4, pp. 1–36, Nov. 2017.

[220] S. Gowda, D. Prajapati, R. Singh, and S. S. Gadre, "False positive analysis of software vulnerabilities using machine learning," in *Proc. IEEE Int. Conf. Cloud Comput. Emerg. Markets (CCEM)*, Nov. 2018, pp. 3–6. [Online]. Available: https://ieeexplore.ieee.org/document/8648633/

[221] Z. Han, X. Li, Z. Xing, H. Liu, and Z. Feng, "Learning to predict severity of software vulnerability using only vulnerability description," in *Proc. IEEE Int. Conf. Softw. Maintenance Evol. (ICSME)*, Sep. 2017, pp. 125–136.

[222] F. Huch, M. Golagha, A. Petrovska, and A. Krauss, "Machine learning-based run-time anomaly detection in software systems: An industrial evaluation," in *Proc. IEEE Workshop Mach. Learn. Techn. Softw. Qual. Eval. (MaLTeSQuE)*, Mar. 2018, pp. 13–18.

[223] G. Jie, K. Xiao-Hui, and L. Qiang, "Survey on software vulnerability analysis method based on machine learning," in *Proc. IEEE 1st Int. Conf. Data Sci. Cyberspace (DSC)*, Jun. 2016, pp. 642–647. [Online]. Available: http://ieeexplore.ieee.org/document/7866201/

[224] J. Kronjee, A. Hommersom, and H. Vranken, "Discovering software vulnerabilities using data-flow analysis and machine learning," in *Proc. 13th Int. Conf. Availability, Rel. Secur.*, Aug. 2018, pp. 1–10.

[225] A. Kuznetsov, Y. Yeromin, O. Shapoval, K. Chernov, M. Popova, and K. Serdukov, "Automated software vulnerability testing using deep learning methods," in *Proc. IEEE 2nd Ukraine Conf. Electr. Comput. Eng. (UKRCON)*, Jul. 2019, pp. 837–841.

[226] M. N. Khalid, H. Farooq, M. Iqbal, M. T. Alam, and K. Rasheed, "Predicting web vulnerabilities in web applications based on machine learning muhammad," in *Proc. Int. Conf. Intell. Technol. Appl.*, vol. 932. Singapore: Springer, 2019, pp. 496–510, doi: 10.1007/978-981-13-6052-7.

[227] W. Niu, X. Zhang, X. Du, L. Zhao, R. Cao, and M. Guizani, "A deep learning based static taint analysis approach for IoT software vulnerability location," *Measurement*, vol. 152, Feb. 2020, Art. no. 107139, doi: 10.1016/j.measurement.2019.107139.

[228] S. Ognawala, R. N. Amato, A. Pretschner, and P. Kulkarni, "Automatically assessing vulnerabilities discovered by compositional analysis," in *Proc. 1st Int. Workshop Mach. Learn. Softw. Eng. Symbiosis*, Sep. 2018, pp. 16–25. [Online]. Available: http://arxiv.org/abs/1807.09160

[229] T. A. Tang, L. Mhamdi, D. McLernon, S. A. R. Zaidi, and M. Ghogho, "Deep learning approach for network intrusion detection in software defined networking," in *Proc. Int. Conf. Wireless Netw. Mobile Commun. (WINCOM)*, Oct. 2016, pp. 1–6.

[230] E. Rezende, G. Ruppert, T. Carvalho, F. Ramos, and P. de Geus, "Malicious software classification using transfer learning of ResNet-50 deep neural network," in *Proc. 16th IEEE Int. Conf. Mach. Learn. Appl. (ICMLA)*, Dec. 2017, pp. 1011–1014.

[231] G. Yan, J. Lu, Z. Shu, and Y. Kucuk, "ExploitMeter: Combining fuzzing with machine learning for automated evaluation of software exploitability," in *Proc. IEEE Symp. Privacy-Aware Comput. (PAC)*, Aug. 2017, pp. 164–175.

[232] M. Zagane, M. K. Abdi, and M. Alenezi, "Deep learning for software vulnerabilities detection using code metrics," *IEEE Access*, vol. 8, pp. 74562–74570, 2020.

[233] F. Howar, K. Meinke, and A. Rausch, "Learning systems: Machine-learning in software products and learning-based analysis of software systems," in *Proc. Int. Symp. Leveraging Appl. Formal Methods*, vol. 1, 2016, pp. 651–654.

[234] H. K. Dam, T. Tran, and A. Ghose, "Explainable software analytics," in *Proc. 40th Int. Conf. Softw. Eng., New Ideas Emerg. Results*, May 2018, pp. 53–56.

[235] C. Kaltenecker, A. Grebhahn, N. Siegmund, and S. Apel, "The interplay of sampling and machine learning for software performance prediction," *IEEE Softw.*, vol. 37, no. 4, pp. 58–66, Jul. 2020.

[236] H.-I. Lim, "Applying code vectors for presenting software features in machine learning," in *Proc. IEEE 42nd Annu. Comput. Softw. Appl. Conf. (COMPSAC)*, Jul. 2018, pp. 803–804.

[237] Q. Liu, X. Li, H. Zhu, and H. Fan, "Acquisition of open source software project maturity based on time series machine learning," in *Proc. 10th Int. Symp. Comput. Intell. Design (ISCID)*, Dec. 2017, pp. 10–13.

[238] M. H. Moghadam, "Machine learning-assisted performance testing," in *Proc. 27th ACM Joint Meeting Eur. Softw. Eng. Conf. Symp. Found. Softw. Eng.*, Aug. 2019, pp. 12–14.

[239] F. Quin, D. Weyns, T. Bamelis, S. S. Buttar, and S. Michiels, "Efficient analysis of large adaptation spaces in self-adaptive systems using machine learning," in *Proc. IEEE/ACM 14th Int. Symp. Softw. Eng. Adapt. Self-Manag. Syst. (SEAMS)*, May 2019, pp. 1–12.

[240] G. K. Rajbahadur, S. Wang, Y. Kamei, and A. E. Hassan, "Impact of discretization noise of the dependent variable on machine learning classifiers in software engineering," *IEEE Trans. Softw. Eng.*, vol. 47, no. 7, pp. 1414–1430, Jul. 2021.

[241] S. E. Sahin, K. Karpat, and A. Tosun, "Predicting popularity of open source projects using recurrent neural networks," in *Proc. IFIP Int. Conf. Open Source Syst.*, vol. 556. Cham, Switzerland: Springer, 2019, pp. 80–90, doi: 10.1007/978-3-030-20883-7_8.

[242] R. Tofighi-Shirazi, I.-M. Asavoae, P. Elbaz-Vincent, and T.-H. Le, "Defeating opaque predicates statically through machine learning and binary analysis," in *Proc. 3rd ACM Workshop Softw. Protection (SPRO)*, 2019, pp. 3–14.

[243] H. Washizaki, H. Takeuchi, F. Khomh, N. Natori, T. Doi, and S. Okuda, "Practitioners' insights on machine-learning software engineering design patterns: A preliminary study," in *Proc. IEEE Int. Conf. Softw. Maintenance Evol. (ICSME)*, Sep. 2020, pp. 797–799.

[244] M. T. Harandi and H.-Y. Lee, "Acquiring software design schemas: A machine learning perspective," in *Proc. 6th Annu. Knowl.-Based Softw. Eng. Conf.*, Sep. 1991, pp. 188–197.

[245] H. Lounis and L. Ait-Mehedine, "Machine-learning techniques for software product quality assessment," in *Proc. 4th Int. Conf. Quality Softw. (QSIC)*, Sep. 2004, pp. 102–109.

[246] H. Lounis, T. F. Gayed, and M. Boukadoum, "Machine-learning models for software quality: A compromise between performance and intelligibility machine-learning models for software quality: A compromise between performance and intelligibility," in *Proc. 23rd Int. Conf. Tools Artif. Intell.*, Nov. 2011, pp. 64–67.

[247] H. Sajnani, "Automatic software architecture recovery: A machine learning approach," in *Proc. 20th IEEE Int. Conf. Program Comprehension (ICPC)*, Jun. 2012, pp. 265–268.

[248] M. C. Simpson, Q. Yi, and J. Kalita, "Automatic algorithm selection in computational software using machine learning," in *Proc. 15th IEEE Int. Conf. Mach. Learn. Appl. (ICMLA)*, Dec. 2016, pp. 355–360.

[249] N. Chen, S. C. H. Hoi, and X. Xiao, "Software process evaluation: A machine learning approach," in *Proc. 26th IEEE/ACM Int. Conf. Automated Softw. Eng. (ASE)*, Nov. 2011, pp. 333–342.

[250] C. Lopez-Martin, A. Chavoya, and M. E. Meda-Campana, "A machine learning technique for predicting the productivity of practitioners from individually developed software projects," in *Proc. 15th IEEE/ACIS Int. Conf. Softw. Eng., Artif. Intell., Netw. Parallel/Distrib. Comput. (SNPD)*, Jun. 2014, pp. 1–6.

[251] D. Rughetti, P. Di Sanzo, B. Ciciani, and F. Quaglia, "Machine learning-based self-adjusting concurrency in software transactional memory systems," in *Proc. IEEE 20th Int. Symp. Modeling, Anal. Simulation Comput. Telecommun. Syst.*, Aug. 2012, pp. 278–285.

[252] H. A. Al-Jamimi and M. Ahmed, "Machine learning-based software quality prediction models: State of the art," in *Proc. Int. Conf. Inf. Sci. Appl. (ICISA)*, Jun. 2013, pp. 1–4.

[253] N. Baskiotis, M.-C. Gaudel, and S. Gouraud, "A machine learning approach for statistical software testing," in *Proc. Int. Joint Conf. Artif. Intell. (IJCAI)*, 2006, pp. 2274–2279.

[254] C. Cummins, P. Petoumenos, A. Murray, and H. Leather, "Compiler fuzzing through deep learning," in *Proc. 27th ACM SIGSOFT Int. Symp. Softw. Test. Anal.*, Jul. 2018, pp. 95–105. [Online]. Available: http://dl.acm.org/citation.cfm?doid=3213846.3213848

[255] H. F. Enişer and A. Sen, "Testing service oriented architectures using stateful service visualization via machine learning," in *Proc. 13th Int. Workshop Autom. Softw. Test*, May 2018, pp. 9–15.

[256] J. Guo, Y. Jiang, Y. Zhao, Q. Chen, and J. Sun, "DLFuzz: Differential fuzzing testing of deep learning systems," in *Proc. 26th ACM Joint Eur. Softw. Eng. Conf. Symp. Found. Softw. Eng.*, Oct. 2018, pp. 739–743, doi: 10.1145/3236024.3264835.

[257] U. Kanewala, J. M. Bieman, and A. Ben-Hur, "Predicting metamorphic relations for testing scientific software: A machine learning approach using graph kernels," *Softw. Test., Verification Rel.*, vol. 26, no. 3, pp. 245–269, May 2016, doi: 10.1002/stvr.1594.

[258] Y. Karpov, Y. Smetanin, and L. Karpov, "Adaptation of general software testing concepts to neural networks," *Program. Comput. Softw.*, vol. 44, no. 5, pp. 43–56, 2019.

[259] S. Masuda, K. Ono, T. Yasue, and N. Hosokawa, "A survey of software quality for machine learning applications," in *Proc. IEEE Int. Conf. Softw. Test., Verification Validation Workshops (ICSTW)*, Apr. 2018, pp. 279–284.

[260] M. Nafreen, S. Bhattacharya, and L. Fiondella, "Architecture-based software reliability incorporating fault tolerant machine learning," in *Proc. Annu. Rel. Maintainability Symp. (RAMS)*, Jan. 2020, pp. 1–6.

[261] S. Nakajima, "Quality assurance of machine learning software," in *Proc. 7th Global Conf. Consum. Electron.*, Oct. 2018, pp. 601–604.

[262] V. U. B. Challagulla, F. B. Bastani, R. A. Paul, W.-T. Tsai, and Y. Chen, "A machine learning-based reliability assessment model for critical software systems," in *Proc. 31st Annu. Int. Comput. Softw. Appl. Conf. (COMPSAC)*, vol. 1, Jul. 2007, pp. 79–86.

[263] R. Rana and M. Staron, "Machine learning approach for quality assessment and prediction in large software organizations," in *Proc. 6th IEEE Int. Conf. Softw. Eng. Service Sci. (ICSESS)*, Sep. 2015, pp. 1098–1101.

[264] D. Shanthi, R. K. Mohanty, and G. Narsimha, "Application of machine learning reliability data sets," in *Proc. 2nd Int. Conf. Intell. Comput. Control Syst. (ICICCS)*, 2018, pp. 1472–1474.

[265] J. Wang and C. Zhang, "Software reliability prediction using a deep learning model based on the RNN encoder–decoder," *Rel. Eng. Syst. Saf.*, vol. 170, pp. 73–82, Feb. 2018. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0951832017303538

[266] M. A. de Almeida, H. Lounis, and W. L. Melo, "An investigation on the use of machine learned models for estimating correction costs," in *Proc. 20th Int. Conf. Softw. Eng.*, Apr. 1998, pp. 473–476.

[267] T. J. Cheatham and N. J. Wahl, "Software testing: A machine learning experiment," in *Proc. 23rd Annu. Conf. Comput. Sci.*, Feb. 1995, pp. 135–141.

[268] S. Ramasamy and I. Lakshmanan, "Machine learning approach for software reliability growth modeling with infinite testing effort function," *Math. Problems Eng.*, vol. 2017, pp. 1–6, Jan. 2017.

[269] M. A. Ahmed and H. A. Al-Jamimi, "Machine learning approaches for predicting software maintainability: A fuzzy-based transparent model," *IET Softw.*, vol. 7, no. 6, pp. 317–326, 2013.

[270] S. Elmidaoui, L. Cheikhi, A. Idri, and A. Abran, "Machine learning techniques for software maintainability prediction: Accuracy analysis," *J. Comput. Sci. Technol.*, vol. 35, no. 5, pp. 1147–1174, Oct. 2020.

[271] S. Reddivari and J. Raman, "Software quality prediction: An investigation based on machine learning," in *Proc. IEEE 20th Int. Conf. Inf. Reuse Integr. Data Sci. (IRI)*, Jul. 2019, pp. 115–122.

[272] J. Alonso, J. Torres, J. L. Berral, and R. Gavaldà, "Adaptive on-line software aging prediction based on machine learning," in *Proc. IEEE Int. Conf. Dependable Syst. Netw.*, May 2010, pp. 507–516.

[273] S. Huo, D. Zhao, X. Liu, J. Xiang, Y. Zhong, and H. Yu, "Using machine learning for software aging detection in Android system," in *Proc. 10th Int. Conf. Adv. Comput. Intell. (ICACI)*, Mar. 2018, pp. 741–746.

[274] S. Lee, C. Jung, and S. Pande, "Detecting memory leaks through introspective dynamic behavior modelling using machine learning," in *Proc. 36th Int. Conf. Softw. Eng.*, May 2014, pp. 814–824.

[275] Y. Yan and P. Guo, "A practice guide of software aging prediction in a web server based on machine learning," *China Commun.*, vol. 13, no. 6, pp. 225–235, Jun. 2016.

[276] K. Zhang, X. Wang, J. Ren, and C. Liu, "Efficiency improvement of function point-based software size estimation with deep learning model," *IEEE Access*, vol. 9, pp. 107124–107136, 2021.

[277] V. François-Lavet, P. Henderson, R. Islam, M. G. Bellemare, and J. Pineau, "An introduction to deep reinforcement learning," *Found. Trends Mach. Learn.*, vol. 11, nos. 3–4, 2018.

[278] S. Shafiq, C. Mayr-Dorn, A. Mashkoor, and A. Egyed, "Towards optimal assembly line order sequencing with reinforcement learning: A case study," in *Proc. 25th IEEE Int. Conf. Emerg. Technol. Factory Autom. (ETFA)*, Sep. 2020, pp. 982–989.

[279] A. Barcus and G. Montibeller, "Supporting the allocation of software development work in distributed teams with multi-criteria decision analysis," *Omega*, vol. 36, no. 3, pp. 464–475, Jun. 2008.

[280] P. Achimugu, A. Selamat, R. Ibrahim, and M. N. Mahrin, "A systematic literature review of software requirements prioritization research," *Inf. Softw. Technol.*, vol. 56, no. 6, pp. 568–585, 2014, doi: 10.1016/j.infsof.2014.02.001.

[281] J. Li, A. Al-Emran, and G. Ruhe, "Impact analysis of missing values on the prediction accuracy of analogy-based software effort estimation method Aqua," in *Proc. 1st Int. Symp. Empirical Softw. Eng. Meas. (ESEM)*, Sep. 2007, pp. 126–135.

[282] C. Wohlin, P. Runeson, P. A. da Mota Silveira Neto, E. Engström, I. do Carmo Machado, and E. S. de Almeida, "On the reliability of mapping studies in software engineering," *J. Syst. Softw.*, vol. 86, no. 10, pp. 2594–2610, 2013, doi: 10.1016/j.jss.2013.04.076.

[283] K. Petersen, S. Vakkalanka, and L. Kuzniarz, "Guidelines for conducting systematic mapping studies in software engineering: An update," *Inf. Softw. Technol.*, vol. 64, pp. 1–18, Aug. 2015, doi: 10.1016/j.infsof.2015.03.007.

[284] B. A. Kitchenham, D. Budgen, and O. P. Brereton, "The value of mapping studies—A participant-observer case study," in *Proc. 14th Int. Conf. Eval. Assessment Softw. Eng. (EASE)*, 2010, pp. 1–9.

ATIF MASHKOOR received the master's degree in computer science from the Umeå University, Sweden, the Ph.D. degree in computer science from the University of Lorraine, France. He was the Scientific Head at the Software Competence Center Hagenberg GmbH—the Austrian Center of Excellence in Data and Software Science. He is currently a Senior Research Scientist with the Institute for Software Systems Engineering and the LIT Secure and Correct Systems Laboratory, Johannes Kepler University, Linz, Austria. He studied computational linguistics at the Rovira i Virgili University, Spain.

CHRISTOPH MAYR-DORN received the Dr.Techn./Ph.D. degrees in computer science from TU Wien, in 2009. He was an Area Manager for "Cognitive Robotics and Shopfloors" at the Austrian Center of Excellence *Pro²Future*. From March 2011 to August 2012, he was a Visiting Researcher with the UC at Irvine. He is currently a Senior Researcher with the Institute for Software Systems Engineering and the LIT Secure and Correct Systems Laboratory, Johannes Kepler University, Linz. His research interests include collaborative engineering environments, software process monitoring and mining, change impact assessment, and software engineering of cyber-physical production systems. He was awarded an FWF Schrödinger Mobility Fellowship (Marie Curie Co-funded).
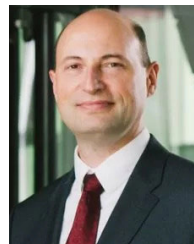
ALEXANDER EGYED (Senior Member, IEEE) received the Ph.D. degree in software engineering from the University of Southern California, USA. He is currently a Full Professor with Johannes Kepler University (JKU), Austria. He is most recognized for his work on software and systems modeling—particularly on the consistency and traceability of models. His work has been published in more than 200 refereed scientific books, journals, conferences, and workshops, with more than 8,000 citations to date. He was recognized among the ten best scholars in software engineering in *Communications of the ACM*, was named an IBM Research Faculty Fellow in recognition to his contributions to consistency checking, received a Recognition of Service Award from the ACM, and various best paper awards. He is a member of the IEEE Computer Society, ACM, and ACM SigSoft.

• • •

SAAD SHAFIQ (Graduate Student Member, IEEE) received the M.Sc. degree in software engineering from the National University of Computer and Emerging Sciences. He is currently pursuing the Ph.D. degree with Johannes Kepler University. He worked as a Software Engineer with ample experience in software development and data science. His research interests include applications of machine learning, requirements engineering, human aspects of software engineering, and empirical software engineering.