IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received March 15, 2022, accepted April 15, 2022, date of publication May 10, 2022, date of current version May 17, 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3174115

Systematic Mapping: Artificial Intelligence Techniques in Software Engineering

HAZRINA SOFIAN[1], (Senior Member, IEEE), NUR ARZILAWATI MD YUNUS[2], AND RODINA AHMAD[1]

[1]Department of Software Engineering, Faculty of Computer Science and Information Technology, University of Malaya, Kuala Lumpur 50603, Malaysia
[2]Faculty of Business and Technology, University of Cyberjaya, Selangor, Cyberjaya 63000, Malaysia

Corresponding author: Rodina Ahmad (rodina@um.edu.my)

This work was supported by the Universiti Malaya under Universiti Grant GPF097A-2020.

ABSTRACT Artificial Intelligence (AI) has become a core feature of today's real-world applications, making it a trending topic within the software engineering (SE) community. The rise in the availability of AI techniques encompasses the capability to make rapid, automated, impactful decisions and predictions, leading to the adoption of AI techniques in SE. With industry revolution 4.0, the role of software engineering has become critical for developing productive, efficient, and quality software. Thus, there is a major need for AI techniques to be applied to enhance and improve the critical activities within the software engineering phases. Software is developed through intelligent software engineering phases. This paper concerns a systematic mapping study that aimed to characterize the publication landscape of AI techniques in software engineering. Gaps are identified and discussed by mapping these AI techniques against the SE phases to which they contributed. Many systematic mapping review papers have been produced only for a specific AI technique or a specific SE phase or activity. Hence, to our best of knowledge within the last decade, there is no systematic mapping review that has fully explored the overall trends in AI techniques and their application to all SE phases.

INDEX TERMS Artificial intelligence, machine learning, deep learning, data mining, software engineering, requirements engineering, analysis and design, software development, software testing, software maintenance, software deployment.

I. INTRODUCTION

As software products become pervasive in all areas of society, the productive building of high-quality software has become crucial to the software industry. The rise of artificial intelligence (AI) applications is potentially a game-changer in improving Software Engineering (SE) phases to ensure higher-quality software, accelerate productivity, and increase project success rates. AI has the capability to assist software teams in many aspects, from automating certain activities in an SE phase to providing project analytics and actionable recommendations, and even making decisions [1]. AI techniques can support software engineers by detecting parts of the SE phases that are more likely to contain vulnerabilities and raising alerts about these issues. Such techniques can help to prioritize efforts and optimize inspection and testing costs. They aim to increase the likelihood of finding vulnerabilities and reduce the time required for software engineers to discover these vulnerabilities. SE phases involve various activities that range across all the stages of the Software Development Lifecycle (SDLC) phases. AI techniques like machine learning (ML), heuristic algorithms (HA), deep learning (DL), data mining (DM), data analytics (DA), and natural language processing (NLP) have been widely explored in the SE phases. As software grows in size, its complexity increases, along with the time and cost required for its overall construction. Extensive data is generated from all the SDLC stages. This data varies between the planning, requirements engineering, design, system development, testing, deployment, training, and maintenance phases [2].

Considering the growing interest in AI techniques in the SE domain, this systematic mapping study attempts to bridge a gap by extensively analyzing the relevant studies published between 2015 and 2021. The choice of the duration is because

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

we saw an escalation of the use of AI techniques within this period of time. The aim is to understand and analyze the AI techniques, SE phases, evaluation metrics, contribution of AI techniques, trends, and demographics of the primary studies. The remainder of this paper is organized as follows: Section II provides works related to reviews of AI in SE. Section III describes the research method used in the current research. Section IV discusses the results and their implications. Finally, Section V presents the concluding remarks.

II. RELATED WORKS

This section aims to highlight review papers published in the AI research domain, emphasizing the need to assess their contributions to the research domain.

Machine Learning (ML) has been widely investigated for building prediction models, estimating software development efforts, and improving the accuracy of other estimation techniques. Ahmad *et al.* [3] reported comprehensively on the planning, conducting, and execution phases of work using ML techniques to recognize diverse software requirements. This study calls for effective collaboration ventures between requirements engineering (RE) and ML researchers or practitioners to tackle the open challenges confronted when developing real-world applications in the broad field of RE. Alsolai and Roper [4] provided a review of software maintainability prediction. This paper analyzes the measurements, metrics, datasets, evaluation measures, ML problems, individual prediction models, and ensemble prediction models employed in the field of software maintainability prediction. Meanwhile, Li *et al.,* [5] reviewed unsupervised software defect prediction primary studies, providing researchers with an indication of the diversity of unsupervised learning techniques used in software defect prediction. Malhotra [6] and Pandey *et al.* [7] performed a systematic literature review to investigate the performance of ML techniques in terms of software fault prediction. The authors summarized the datasets, reduction/selection techniques, ML techniques, software features, and performance measures before investigating them. The performances of these predicted models using ML techniques were compared with other ML techniques. The analysis showed that the ML-based models outperformed the classical statistical methods. Azeem *et al.* [8] reported a review of the use of ML techniques for code smell detection. This paper highlighted a number of limitations in the existing studies, as well as the open issues that had to be addressed in future research.

Binkhonain and Zhao [9] reported a systematic review of selected ML approaches used for identifying and classifying non-functional requirements in requirements documents. This paper presents the results of recent advances in RE research. More significantly, using ML for RE provides exciting opportunities to develop novel expert and intelligent systems to support RE tasks and processes.

The systematic mapping by Durelli *et al.* [10] focused on surveying research efforts that were based on using ML algorithms to support software testing. Most solutions proposed in the selected studies scale well and show that some problems currently faced by the software-testing industry could be addressed using ML-based solutions. Nevertheless, few primary studies evaluated their ML-based solutions in industrial settings or using industry-grade software.

Data mining is the process of uncovering patterns and other valuable information from large datasets. With the evolution of data warehousing technology and the growth of big data, the adoption of data mining techniques has rapidly accelerated over the last two decades, assisting companies by transforming their raw data into useful knowledge. El- Masri *et al.* [11] reported a systematic review of the literature on Automated Log Abstraction Techniques (ALATs). The authors proposed a quality model with seven industry-relevant quality aspects for evaluating ALATs. Researchers can use this model and the recommendations to learn about the state-of-the-art ALATs, understand research gaps, enhance existing ALATs, and/or develop new ones. Software engineers can use the model and recommendations to understand the advantages and limitations of existing ALATs and identify which ones best fit their needs. Nayebi and Abran [12] concluded that spam or fake review detection is one of the major problems in the mobile application domain. In addition to spam reviews, there are various kinds of user reviews, some of which include no useful data for information extraction. Hence, it is necessary to merge multiple criteria to not only identify suspicious reviews but also differentiate useful reviews from others so that reviews complying with the usefulness criteria can be processed for information extraction.

Villamizar *et al.,* [13] conducted a study that aimed to characterize the publication landscape of RE for ML-based systems, outlining the research contributions and contemporary gaps for future research. These contributions comprise analyses, approaches, checklists and guidelines, quality models, and taxonomies. Hence, two main contributions were made to this research: (i) mapping relevant knowledge about the current state of RE for ML and (ii) helping to identify points that still require further investigation.

The existing systematic reviews have focused on examining one or several AI technique(s) that have been used in a particular SE phase(s). Hence, there is a foremost need to perform a systematic mapping review examining the major AI techniques in all the main SE phases.

III. RESEARCH METHOD

Systematic mapping or scoping studies are conducted to provide an overview of a research domain through classification. These studies mainly explore the existing literature to investigate the coverage of multiple topics, the frequency of publications, the research trends, and the publication venues where relevant studies have been published [14]. The systematic mapping in the current study mainly follows the guidelines suggested by Petersen *et al.* [15]. According to the guidelines for systematic mapping studies in SE [15], the essential process steps of the current systematic mapping study were

defining the research questions, searching for relevant papers, screening the papers, keywording the abstracts, extracting the data, and mapping, as shown in Fig. 1. Each process step has an outcome and the outcome of the complete process is the systematic map, which is explained as follows [15]:

Definition of Research Questions (Research Scope) - The primary goal of a systematic mapping study is to provide an overview of a research area and identify the quantity and type of research and results available within this area.

Conduct Search for Primary Studies (All Papers) - Primary studies were identified using search strings on scientific databases or browsing manually through relevant conference proceedings or journal publications.

Screening of Papers for Inclusion and Exclusion (Relevant Papers) - Inclusion and exclusion criteria were used to exclude studies that were not relevant for answering the research questions.

Keywording of Abstracts (Classification Scheme) - Keywording is a way to reduce the time needed to develop the classification scheme and ensure that the method considers the existing studies.

Data Extraction and Mapping of Studies (Systematic Map) - Once the classification scheme was in place, the relevant articles were sorted into the scheme, i.e., the actual data extraction took place.

A. RESEARCH QUESTIONS

The primary RQ of this systematic mapping study was: 'What AI techniques are used in SE?' This primary question was divided into six RQs. Table 1 lists the formulated RQs along with the rationale behind each RQ.

B. DATA SOURCES

Six electronic databases were considered as primary data sources for potentially relevant studies. Google Scholar was not included in the list due to its low-precision results and the overlapping of results from other data sources. The electronic databases used in the search process are listed in Table 2.

C. SEARCH TERMS

Identifying the relevant search terms is essential for an adequate search of the relevant studies. Kitchenham *et al.* [16] suggested population, intervention, comparison, and outcome (PICO) viewpoints in this regard. These viewpoints have been extensively used by several SLRs. Here, the relevant PICO terms are listed:

Population: primary studies in software engineering?

Intervention: AI techniques?

Comparison: Techniques, Contribution, Dataset, Performance metric, and SE phases

Outcome: Classification of the types and combination of AI techniques applied in SE phases.

On the basis of the PICO structure, a generic search string was constructed to maintain the consistency of the search across multiple databases:

("Artificial intelligence" OR "Artificial intelligence techniques" OR "Machine Learning" OR "Deep learning" OR "Data mining" OR "Big data" OR "Swarm intelligence" OR "Sentiment analysis" OR "Predictive models" OR "Analytics models" OR "Heuristic Algorithm" OR "Data Driven" OR "Natural Language Processing") AND ("Software engineering" OR "Requirements Engineering" OR "Software Planning" OR "Software Analysis" OR "Software Design" OR "Software Development" OR "Software Testing" OR "Software defect prediction" OR "Software Deployment" OR "Software Training" OR "Software Maintenance").

D. INCLUSION AND EXCLUSION CRITERIA

Inclusion and exclusion criteria were used to select and deselect studies from the data sources to answer the RQs in this systematic mapping study. These criteria were applied to all the studies retrieved during the different phases of the study selection procedure (see Table 3.) Early cited articles were also included, provided the full text was available.

E. SOFTWARE ENGINEERING PROCESS

The authors introduced several SE phases, as illustrated in Fig. 2. These SE phases included features and observations from existing studies and consisted of eight phases: (i) planning, (ii) requirements engineering, (iii) design, (iv) system development, (v) testing, (vi) deployment, (vii) training, and (viii) maintenance.

1) PLANNING

The planning phase is when the project plan is developed and involves identifying, prioritizing, and assigning the tasks and resources required to build the structure for a project.

2) REQUIREMENTS ENGINEERING

This is the second phase, in which each set of relevant information is elicited, analyzed, prioritized, and negotiated. The next phase of RE involves a detailed definition of the requirements, documenting these requirements, and getting verification from the stakeholders.

3) DESIGN

This is the third phase and is concerned with the whole structure of the future project to be implemented. The software requirement specification is used as the input to design the architecture,database and user interface of the software product being developed.

4) SYSTEM DEVELOPMENT

The fourth step consists of the coding that builds the software. It incorporates all elements such as classes, files, infrastructure, and executable task. In this phase, the actual development of the product starts according to the designed architecture.

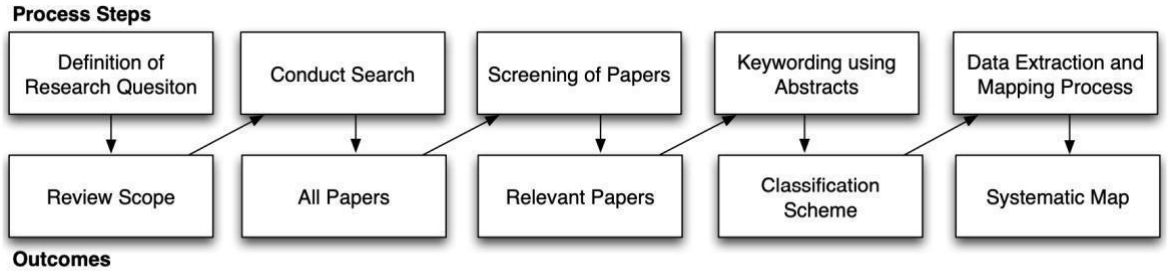

FIGURE 1. Systematic mapping process [15].

TABLE 1. Research questions.

RQ No.	Research questions	Motivation
RQ1	In which phases of SE have AI techniques been heavily applied?	To identify the SE phases in which AI techniques have been heavily applied.
RQ2	What types of AI techniques have been used in SE? (i.e., ML, DL, DM, HA)	To identify the most common AI techniques that have been used in SE.
RQ3	What is the contribution of each AI technique in SE?	To distinguish the contributions made by applying the AI approach and developing new techniques and hybrid AI techniques, as proposed in the primary studies.
RQ4	What are the most frequently used evaluation metrics?	To highlight the most frequently used evaluation metrics, based on the AI techniques used in software engineering.
RQ5	What are the trends and directions of the application of AI techniques in SE?	To highlight the current AI capabilities used in SE. To discuss the future direction of AI techniques in SE.
RQ6	What are the demographics of the primary studies?	To highlight the distribution of primary studies based on the type, year, and venue of publication.

5) TESTING

The testing involves the verification and correction of any defective parts of the code. All are thoroughly tested and retested if necessary until all the problems are resolved.

6) DEPLOYMENT

In the deployment phase, the software/subsystem/components are placed and executed on the required physical hardware nodes (e.g., mobile, server, or desktop). The project is then made available to the end-users.

7) TRAINING

The training activities perform a detailed audience and task analysis through interaction with the proposed users of the system, the key client representatives, whereby the training increases knowledge and usability.

8) MAINTENANCE

The maintenance phase follows the entire life cycle of the software. If users or developers find a problem, it may be fixed or corrected ahead of the next release, depending on how serious it is.

IV. RESULTS AND DISCUSSION

In this section, all the RQs are answered by analyzing the results extracted from the collection of primary studies.

A. RQ1: IN WHICH PHASES OF SE HAVE AI TECHNIQUES BEEN HEAVILY APPLIED?

Based on the current analysis of the selected primary papers, eight SE phases were identified. In total, 23 of the papers implemented AI techniques in the requirements engineering phases, 16 in the system development phases, 13 in the testing phases, and 9 in the maintenance phases, as depicted in Fig. 3. Meanwhile, the remaining 3, 3, 2 and 2 implemented AI techniques in planning, deployment, design, and training phases, respectively.

Recently, many RE methodologies have focused on deriving the requirements for resolving customer problems using AI techniques [17]. Subsequently, some approaches have explored how to integrate ML with the RE phase, as shown in

TABLE 2. Electronic database.

Database name	Link
Taylor and Francis	https://taylorandfrancis.com
MDPI	https://www.mdpi.com
IEEE Xplore	https://ieeexplore.ieee.org/Xplore/home.jsp
Science Direct	https://www.sciencedirect.com
Springer Link	https://link.springer.com
Wiley	https://onlinelibrary.wiley.com

TABLE 3. Inclusion and exclusion criteria.

Inclusion criteria	
IC1	Articles that are peer-reviewed
IC2	Articles providing the SE phases and integrating artificial intelligence techniques
IC3	Inclusion of the most recent article in the case of multiple studies on same theme
IC4	Articles published from 2015 to 2021
Exclusion criteria	
EC1	Articles that do not meet the inclusion criteria
EC2	Articles that are only available in a form of an abstract or presentation
EC3	Studies in languages other than English
EC4	Studies with no validation of the proposed techniques or validation solely through expert opinion
EC5	Articles providing unclear results or findings

Table 4. The RE phase adopts ML the most, compared to the other SE phases. Specifically, the analytics design view and the data preparation view may be ideally suited to support activities within the RE phase. From the analysis shown in Table 4, the system development phase has also heavily adopted AI techniques. Based on the current analysis of previous research, the authors found that ML is the AI technique that is increasingly being adopted in the system development phases. The testing phase involves probing into the behavior of software systems to uncover faults. Most testing activities are complex, resource-consuming, and costly. Hence, many practical strategies have been adopted to circumvent these issues. As Table 4 highlights, the authors found that ML has also been widely used in the testing phase. This study shows that ML is most commonly used to automate software testing activities and help researchers to identify critical test plans for assisting in saving resources.

B. RQ2: WHAT TYPES OF AI TECHNIQUES HAVE BEEN USED IN SE? (I.E., MACHINE LEARNING, DEEP LEARNING, DATA MINING, HEURISTIC ALGORITHM)

Along with the flourishing of AI techniques in modern computing, many researchers and studies are investing AI in their domain. Referring to Table 4, eight AI techniques have been used extensively to facilitate activities in SE: Machine Learning (ML), Heuristic Algorithm (HA), Deep Learning (DL), Data Driven (DD), Machine Learning (ML) + Natural Language Processing (NLP), Machine Learning (ML) + Data Driven (DD), Machine Learning (ML)+ Heuristic Algorithm(HA), and Deep Learning (DL) + Data-Driven (DD). Each AI technique has its own AI capabilities (refer to section IV - E). The current analysis reveals that ML in SE has recently received increased attention. There has been a growing interest in applying ML to automate various SE phases. These findings show that ML has been applied to

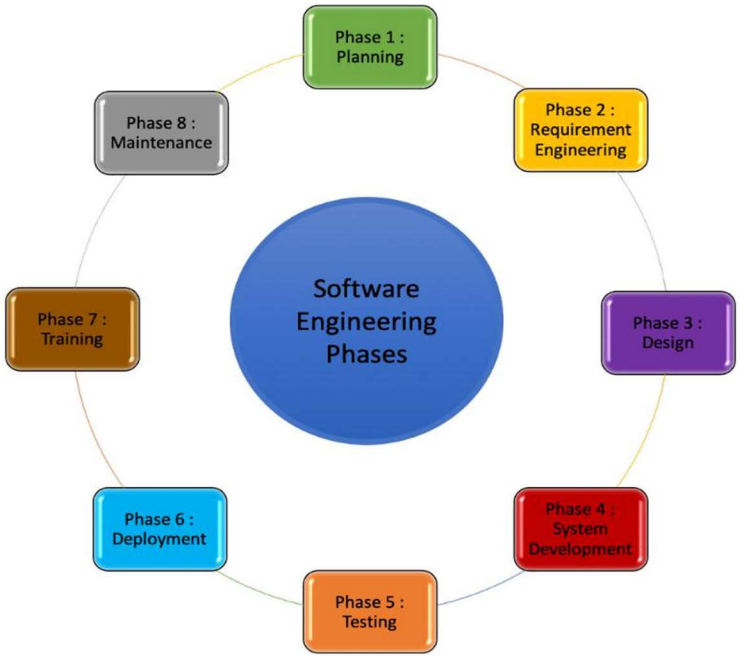

FIGURE 2. Software engineering phases.

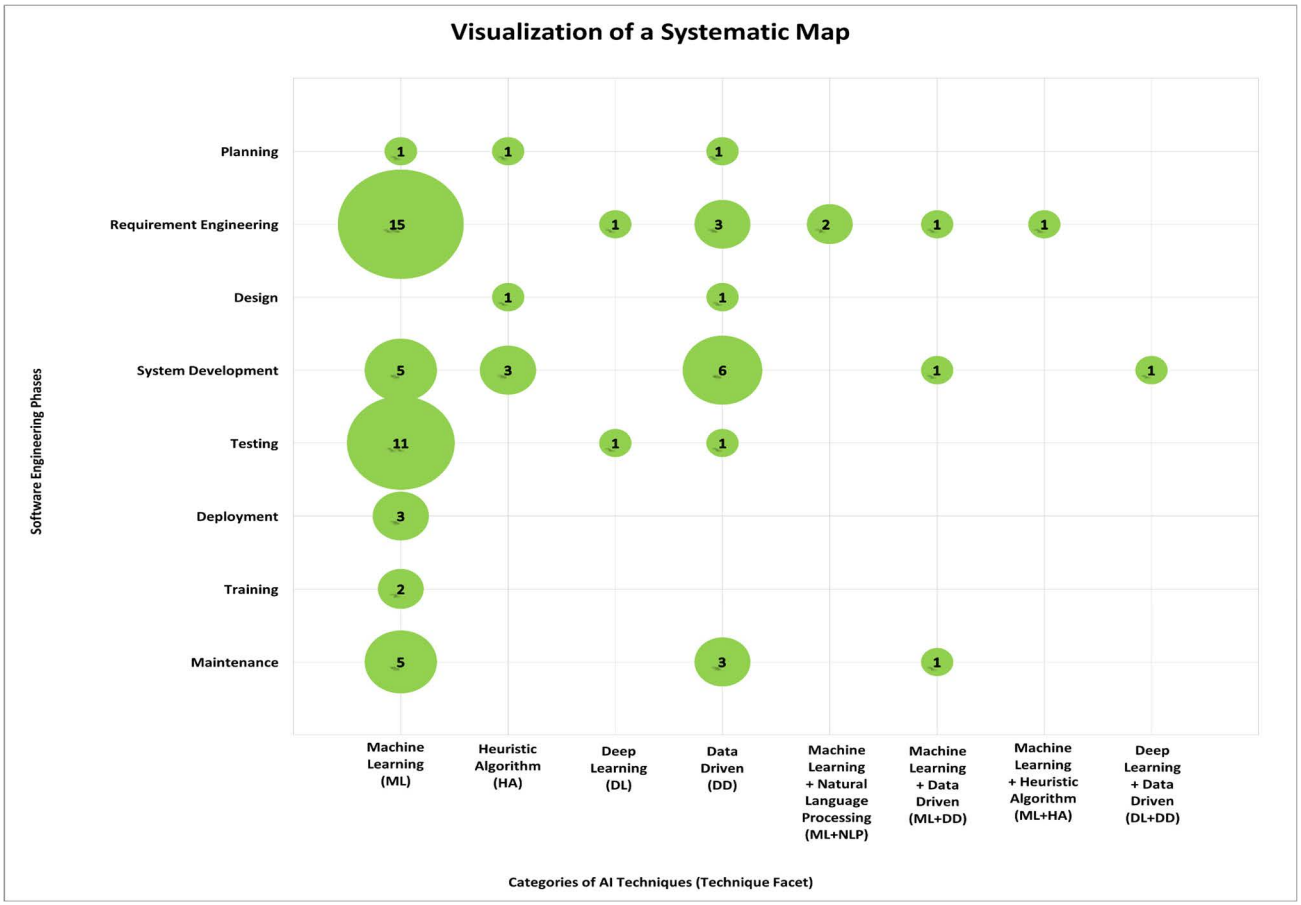

FIGURE 3. Visualization of a systematic map for AI techniques in software engineering phases.

TABLE 4. Paper distribution in software engineering phases.

Software Engineering Phases	Categories of AI Techniques								Number of papers
	ML	HA	DL	DD	ML+NLP	ML+DD	ML+HA	DL+DD	
Planning	PS1	PS25		PS36					3
Requirements engineering	PS3, PS8, PS9, PS10, PS22, PS29, PS30, PS31, PS32, PS33, PS34, PS41, PS57, PS59, PS60		PS45	PS11, PS51, PS56	PS4, PS5	PS54	PS8		23
Design		PS24		PS36					2
System development	PS2, PS18, PS27, PS42, PS55	PS13, PS14, PS46		PS7, PS19, PS43, PS47, PS50, PS53		PS54		PS49	16
Testing	PS2, PS12, PS15, PS24, PS27, PS28, PS37, PS38, PS39, PS40, PS52		PS17	PS23					13
Deployment	PS20, PS22, PS26								3
Training	PS1, PS16								2
Maintenance	PS1, PS21, PS37, PS52, PS58			PS23 PS48 PS50		PS44			9

address a myriad of SE problems and it has attracted greater interest among researchers and practitioners than other AI techniques. The current results suggest that interest in applying ML in the SE phases has risen considerably in recent years. Data-driven techniques were reported as the emerging AI techniques in SE. Data-driven techniques can identify and evaluate which decision-modeling techniques would help in making important decisions. Data-driven techniques usually explore the potential of predictive analytics offered by AI techniques, which enables AI-based decision making.

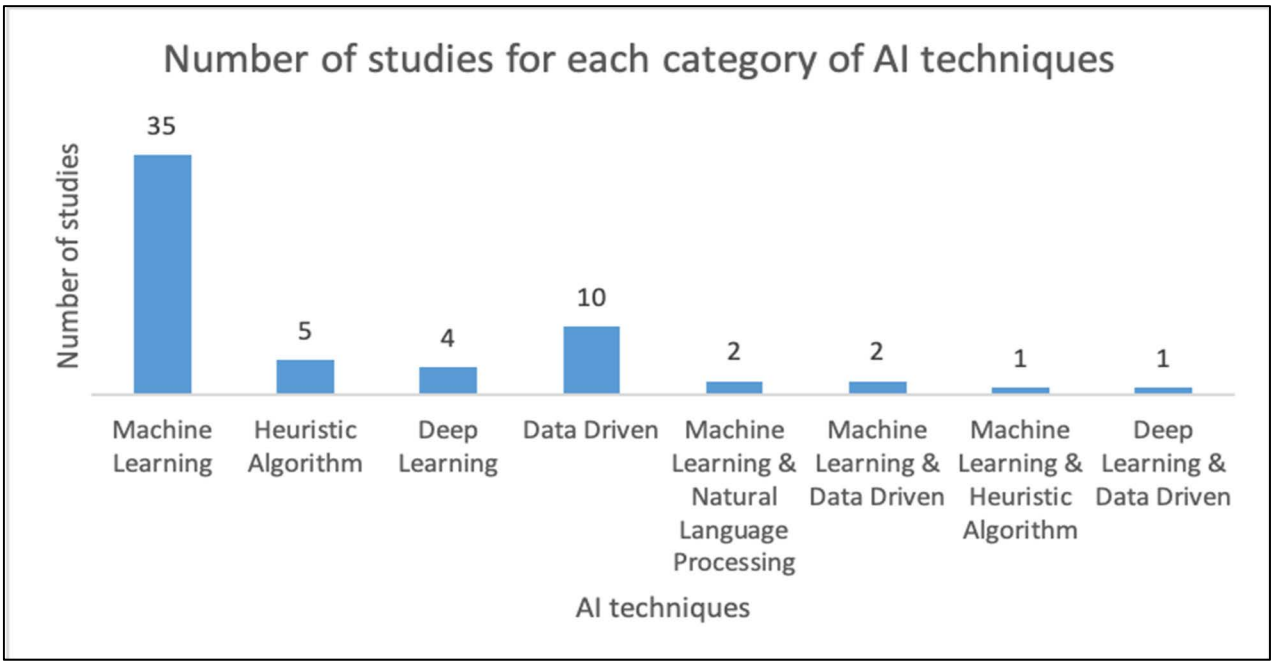

FIGURE 4. Number of studies for each category of AI techniques.

TABLE 5. Artificial intelligence techniques in the primary studies.

Artificial Intelligence Techniques	Primary Studies	Number of papers
ML	PS1, PS2, PS3, PS8, PS9, PS10,PS12,PS15, PS16,	35
	PS18, PS20, PS21, PS22, PS26, PS27, PS28, PS29,	
	PS30, PS31, PS32, PS33,PS34, PS35, PS37, PS38,	
	PS39, PS40, PS41, PS42, PS52, PS55, PS57, PS58,	
	PS59, PS60	
HA	PS13, PS14, PS24, PS25, PS46	5
DL	PS17, PS45, PS48, PS50	4
DD	PS7, PS11, PS19, PS23, PS36, PS43, PS47, PS51,	10
	PS53, PS56	
ML & NLP	PS4, PS5	2
ML & DD	PS44, PS54	2
ML & HA	PS6	1
DL & DD	PS49	1

The heuristic algorithm is another important AI technique that has been used in SE. This is designed to solve a problem faster and more efficiently than the traditional methods by sacrificing optimality, accuracy, precision, or completeness for speed. Heuristic algorithms (HA) are used to solve SE problems, a class of decision problems. A HA can produce a solution individually or be used to provide a good baseline and then supplemented with optimization algorithms.

C. RQ3: WHAT IS THE CONTRIBUTION OF EACH AI TECHNIQUE IN SE?

As shown in Table 6, ML is the main AI technique that is often used in SE. ML can be used as a stand-alone technique or be combined with other AI techniques to produce hybrid techniques, which can be more efficient in solving SE problems. ML can also perform pattern recognition and statistical inference for learning, in order to improve its performance when carrying out a task. Most researchers in SE have applied

TABLE 6. Application, improvement and development of artificial intelligence techniques in the primary studies.

Application / Improvement / Development	AI Techniques	Primary Studies	Number of papers
1. Application of Single AI technique	Apply ML	PS2, PS3, PS8, PS9, PS10, PS12, PS16, PS18, PS20, PS21, PS22, PS27,	32
		PS28, PS29, PS30, PS31, PS32, PS33, PS34, PS35, PS37, PS38, PS39,	
		PS40, PS41, PS42, PS52, PS55, PS57, PS58, PS59, PS60	
	Apply DL	PS17, PS45	2
	Apply DD	PS7, PS11, PS19, PS23, PS36, PS43, PS47, PS48, PS50, PS51, PS53,	12
		PS56	
	Apply HA	PS13, PS14, PS24, PS25, PS46	5
2. Application of hybrid of AI techniques	Apply ML & NLP	PS4, PS5	2
	Apply HA & ML	PS6	1
	Apply DD & ML	PS44, PS54	2
	Apply DD & DL	PS49	1
	Apply hybrid ML	PS26	1
3. Improvement of AI techniques and model	Improve ML	PS15	1
4. Development of new AI technique or model	Develop new ML	PS1	1

ML for predicting the coding intricacy [18], source code mining [19], high level requests [20], the delay of issues [21], software defection [22]–[24], and software request classification [25]–[27]. On the other hand, ML can also classify a textual conversation [28] and improve requirements traceability recovery [29]–[31], software quality [32], bug fixing [33], [34], security control [35], maintainability [36], and vulnerability prediction [37]. A data-driven model is based on the analysis of the data about a specific system. Data-driven techniques have been applied for domain validation [38], speed delivery [39], defect prediction [40], mining treatment [41], application classification [42], decision making [43], error correcting [44], security enhancement [45], delivery prediction [46], and enhancing requirements reusability [47].

HAs are often used in AI to get a computer to find an approximate solution instead of an exact one [48]. Previous researchers have applied HA techniques in AI to reduce the efforts of SE updates [49], as well as predict continuous failure [50] and software reliability [51]. HAs also help to solve agile development problems with swarm intelligence algorithms [52]. ML aims to provide increasing levels of automation in the knowledge-engineering process, replacing much time-consuming human activity with automatic techniques [53]. For this reason, ML has been successfully applied in many areas, such as fraud detection, speech and image recognition, and NLP [54]. Anh Do *et al.* and Khan *et al.* applied ML and NLP to automatically classify and capture creative requests [55] and provide resource knowledge acquisition from online forums [38]. By learning from previous computations and extracting regularities from massive databases, ML can help to produce reliable and repeatable decisions. Marcen *et al.* applied HA and ML in developing traceability models [56].

The main concept behind the data-driven model is to find relationships between system state variables without explicit knowledge of the system's physical behavior [53]. Papamichail *et al.* and Fabian *et al.* applied data-driven approaches and ML to software sustainability [57] and mining validity [58]. Malhotra and Khanna applied hybrid ML for software change prediction [59]. Meanwhile, Siavvas *et al.* and Ding *et al.* developed new forms of ML to increase the accuracy when predicting software security risk [60] and improve ML software defect prediction [49]. The evolution of artificial neural networks (ANNs) involves increasingly deep neural network architectures with improved learning capabilities, summarized as DL [54]. Ranjan Bal *et al.* and Morakot *et al.* applied DL for imbalanced learning in software fault prediction [61] and estimating story points [62].

Overall, Table 5 has shown the classified studies according to the type of AI techniques that have been used. We also observed that some of the AI techniques have been singly applied and some are used in combination or hybrid with

other techniques. Even more, some of the research work improves on the existing AI techniques before using them and some even develop new models. The classification has been detailed in Table 6.

D. RQ4: WHAT ARE THE MOST FREQUENTLY USED EVALUATION METRICS?

Evaluation metrics are used to measure the quality of statistical or ML models. Many different types of evaluation metrics are available to test an ML model. It is highly important to use multiple evaluation metrics because a model may perform well using one measurement from one evaluation metric but poorly using another measurement from another evaluation metric. Using the appropriate evaluation metrics is critical in ensuring that the ML model is operating correctly and optimally. Quantitative evaluation is a method that yields numerical indices gathered primarily from objective methods of data collection, systematic and controlled observation, and a prescribed research design. In particular, the evaluation metric must be specified. In ML systems, the precision, recall, F1-score, F-measure, accuracy, and area under curve (AUC) were considered the highest forms of evaluation used by researchers in this study, as shown in Table 7.

E. RQ5: WHAT ARE THE TRENDS AND DIRECTIONS OF THE APPLICATION OF AI TECHNIQUES IN SE?

Based on the study analysis, we perceive the current trend in SE is the exploitation of five AI capabilities: prediction, assisting in decision making, automation of SE activities, improving performance metric values and hybridization of automation and improvement for performance metrics. Each AI capability is enabled by the eight AI techniques mentioned in Section IV-B. The following sections explain the AI capabilities, which were arranged based on which were most frequently adopted during SE activities.

AI Capability 1 (Prediction): Predicting refers to anticipating what will happen in the future [1]. For example, the testing phase predicts the existence of vulnerability. Most software vulnerabilities are caused by coding, design errors [63], and software delivery speed [39]. Therefore, several ML models have been built to evaluate the capacity of the indicators to predict the existence of vulnerabilities in software classes and the delivery speed [63], [39]. The development of prediction models for predicting changes in internal quality attributes using source-code metrics and ML algorithms helps to improve the software quality and reduce the cost of fixing these bugs [33]. Redundant pruning considers the interaction between features, which improves the prediction performance of the proposed algorithm. The grid-search method is used to control true and false minimum support [23]. Genetic programming (GP) adaptation and prediction rules are represented as a combination of metrics and threshold values that should correctly predict, as far as possible, the failed builds extracted from a base of real-world examples [64]. A new software defect prediction method, Pruned Histogram-based Isolation Forest (PHIForest) [49], and the Weighted Regularization Extreme Learning Machine (WR-ELM) model [61] can mitigate some of the side effects caused by an imbalanced training dataset and enhance prediction performance by changing the design or the simple logic of the software. Moustafa *et. al*, on the other hand, proposed ensemble classifiers to solve the problem of class-imbalanced datasets by improving their classification accuracy [24]. Technology-oriented learning, a project-oriented educational environment, can also benefit from ML classifiers to allow the early prediction of low performers, leading to better team–instructor synchronization [65]. Choetkiertikul *et al.* developed an accurate model to predict whether an issue would be at risk of being delayed and, if so, the extent of the delay (risk impact) and the likelihood of the risk occurring [21]. A cooperative learning environment with the aid of technology may be a key revolution in the project-oriented academic environment [66]. Project models can be effectively exploited by practitioners to predict community smells, especially when trained using Random Forest as the classifier [67]. Since many SE organizations do not yet have a large codebase when they most want to be able to predict external quality issues, this generalization may allow organizations to train models on other projects and then use these models to predict the quality issues that may affect their own [68]. Better support for predicting when high-level requirements will be completed can help project managers to make better planning decisions [20]. A predictive model can be built which, after proper validation, will be able to predict future defects in a given project [40], [51], [59], [46]. The maintainability evaluation methodology targets multimedia projects that harness information residing in code hosting facilities [69].

The poor performance of the SE process is due to the different dataset characteristics of source project data and target project data. Effective multi-objective learning techniques in cross-project environments help to address this problem [22]. Requirements quality can be used as a predictive factor for end-system operational performance [70]. Combining features from multiple models did not necessarily result in increased accuracy and sometimes resulted in a degradation in performance [37].

AI Capability 2 (Assisting in Decision Making): Decision making plays an important role in the success of a business or an organization. Based on multiple criteria, decision making is difficult as several available criteria may be applied when making a decision. It is crucial to provide clear guidance for understanding and addressing the difficulties inherent in high-quality AI systems. The current version was constructed in a bottom-up, best-effort manner to identify what was missing from the guidelines or the knowledge of research communities [71]. The proposed Crowd-based Requirements Engineering by Argumentation (CrowdRE-Arg) approach can be utilized by future user forums and other social media platforms to make them more intuitive and user-friendly [72]. An initial human subject evaluation was conducted of the framework using feature descriptions from three application domains and the results demonstrate the framework's ability

to generate creative requirements [55]. Interaction design patterns help in complying with the usability and user experience principles, thus promoting the usage of standards and best practices [27].

Software practitioners face the problem of how to reliably evaluate the performance of defect classifiers to select the best-performing out of several classifiers. The Fuzzy Analytical Hierarchy Process (FAHP) approach will increase software developers' confidence in research outcomes and help them to avoid false conclusions and infer reasonable boundaries [73]. The Decision Tree for small- and medium-sized software and the ELM for large-sized software can be a solution to deliver software in scheduled time [69]. Pickerill *et al.* developed a model that fosters the sharing and reuse of knowledge, components, tools, and concepts to accelerate the instantiation process [54]. Researchers often rely on quick and dirty techniques to curate datasets. The Project History Analysis of Time-Series Method (PHANTOM) improves the existing methods with respect to time analysis and hardware requirements but without sacrificing accuracy [74]. Applications without well-rounded considerations of security problems are becoming increasingly important for a successful product. Developers get useful information according to their demands from two aspects: security requirements for the whole App and those for the given functionality [45].

AI Capability 3 (Automation of SE Activities): Automation has been around for centuries, far longer than humans have had computers. Automation means that machines replicate human tasks. However, AI demands that machines also replicate human thinking. Presently, there are limited approaches for extracting requirement-related information from the community forums. The automated approach automatically identifies the rationale and requirements-related information using ML algorithms from the Reddit forum [38]. The Evolutionary Learning to Rank for Traceability Link Recovery (TLR-EL to R) approach recovers traceability links between the requirements of a software system [75]. A set of useful ML algorithms have been adopted to discover a wide spectrum of traceability links [31]. The semi-automated approach can be classified as a form of Intelligent Computer-Aided Software Engineering (I-CASE) that utilizes standard NLP and ANN approaches, both types of AI, to extract actors and actions from descriptions of use case models' NL requirements [30].

When the data is very large and/or is expressed in terms of a complex model of software projects, interpretation is often complicated. In software analytics, it is possible, useful, and recommended to combine data mining and optimization using data miners that use/ are used by optimizers (DUO) [76]. Watanabe *et al.* proposed an approach to reducing the number of false negatives in the classification process [77]. While based on deep learning, fully end-to-end prediction systems were proposed by Choetkiertikul *et al.* for estimating story points, removing the users from manually designing features of the textual description of issues [62]. Classification models derived from historical data largely align with expert reasoning about the applicability of security controls [35]. A new approach, Active Learning Fingerprint-based Anti-Aliasing (ALFAA), was developed by Amreen *et al.* for correcting identity errors in the SE context and applied in the OpenStack ecosystem [44].

AI Capability 4 (Improving Performance Metric Value): Supporting the improvement process means a project that results in the ability of the region to enhance services in particular areas. Certain threats are accounted for in almost all the studies but some threats have been ignored by most of them. Extracts of remedial actions from the literature, which should be followed by future researchers to provide an effective experimental setup, yield practical and reliable results while developing software prediction models using search-based approaches [78]. The framework proposed by Haider *et al.* used the team foundation server repository for communication and coordination purposes within global software development, a case-based ranking to find their ranking from the previous requirements, and then the J48 classification [79]. Automatic software requirements classification can help developers to document their projects more effectively, minimize reworking, and make the software easier to use and understand [80]. To date, the classification accuracy has not been satisfactory. An ensemble model was produced that enhanced the weighted voting ensemble by using the accuracy per class for each base ensemble to determine the best classification of the functional requirements in six classes to enhance accuracy and availability [26]. SmartSHARK, developed by Trautsch *et al.*, can execute different plug-ins for an arbitrary git-using project [58].

The need for the proper identification of software requirements at an early stage is crucial since continuous changes to initial requirements can lead to faults. Semantic information that allows flexibility and traceability is effective for providing useful recommendations to add to a system or improve already identified requirements [81]. The maintainability of legacy systems has always been a key task for software companies, considering that the inverse requirements were error-free and easily changeable according to customer requirements [36]. Catal and Guldan proposed a model combining the power of five high-performance individual classifiers and using the majority voting aggregation rule. Since this does not rely on a single classifier and evaluates the result of each classifier, its generalization capability is high [47]. The time invested in testing the software during the software cycle can be reduced to datasets that incur changes and errors. The process can be performed efficiently and effectively [47].

AI Capability 5 (Hybrid of Automation of SE Activities and Improvement of Performance Metric Value): More than 45% of the software development effort has been expended on software maintenance for fixing software bugs. Crowdsourcing has been employed to facilitate attribute construction to improve the bug report summarization task [34].

TABLE 7. Performance metrics in the primary studies.

Performance Metric	Primary Studies	Number of Papers
Precision	PS1, PS2, PS3, PS5, PS6, PS9, PS10, PS13, PS16, PS19, PS27, PS30, PS32, PS33, PS35, PS37, PS38, PS40, PS47, PS48, PS49, PS50, PS51, PS59, PS60	25
Recall	PS1, PS2, PS3, PS5, PS6, PS9, PS10, PS13, PS16, PS19, PS27, PS30, PS32, PS33, PS35, PS37, PS39, PS47, PS48, PS49, PS50, PS52, PS59, PS60	24
F-measure	PS1, PS2, PS3, PS6, PS9, PS15, PS16, PS20, PS27, PS40, PS47	11
Accuracy	PS12, PS20, PS30, PS31, PS40, PS41, PS48, PS49, PS55, PS57	10
F-value	PS29, PS33	2
Clarity, novelty, usefulness	PS4	1
F1-score	PS5, PS10, PS13, PS30, PS32, PS33, PS35, PS37, PS38, PS48, PS49, PS52	12
MCC	PS6, PS16	2
Query Validation	PS7	1
Satisfaction level	PS8	1
Absolute residuals, mean absolute residuals, standardize performance	PS11	1
AUC	PS12, PS14, PS15, PS16, PS19, PS28, PS35, PS38, PS39	9
Average absolute error, average relative error	PS17	1
Complexity	PS18, PS20, PS44	3
Inheritance	PS18, PS44	2
Coupling	PS18, PS44	2
Cohesion	PS18, PS44	2
False alarm	PS19, PS26, PS35	3
Sensitivity, specificity	PS21	1
G-mean	PS21, PS26, PS39	3
Balance	PS21, PS26, PS39	3
Feature engineering, model features, general features	PS22	1
Has Defects, K-Fold validation	PS23	1
Reusability, component interaction, component dependency	PS24	1

TABLE 7. *(Continued.)* **Performance metrics in the primary studies.**

Conclusion validity threat, Internal validity threat, Construct validity threat	PS25	1
Linguistic quality defect	PS29	1
True positive	PS29, PS40	2
False positive	PS29, PS40	2
Validity	PS34	1
Time performance	PS36	1
Effort size relationship	PS42	1
Data and knowledge, tools and methods, concept	PS43	1
Median absolute error, mean absolute error, standard accuracy	PS45	1
Next release problem, risk, software design, software cost estimation, software effort estimation	PS46	1
Threshold	PS47	1
Split, lump	PS50	1
Feature aggregation	PS53	1
Execution time	PS54	1
Recommended requirement	PS56	1
Correctly predicted error, software maintainability	PS58	1

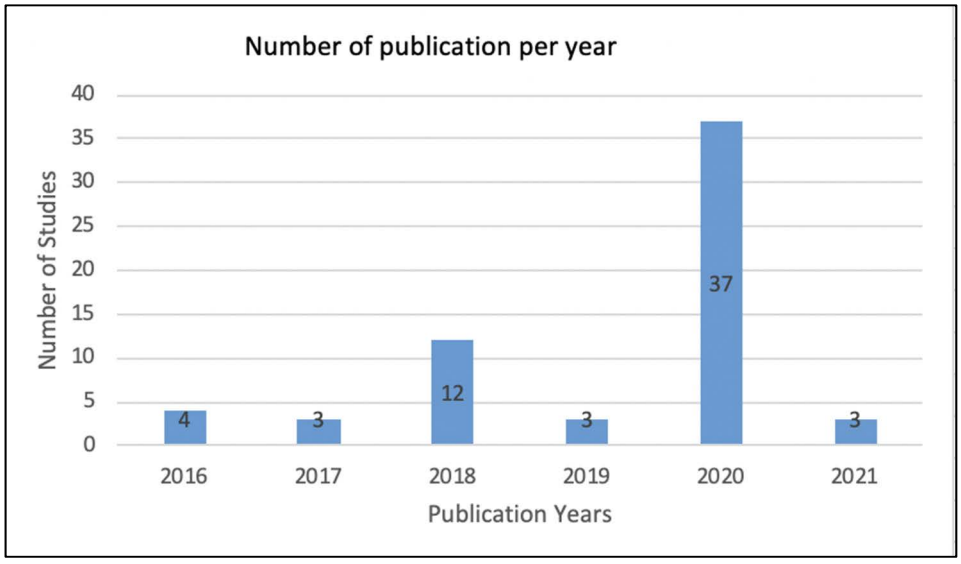

FIGURE 5. Number of publications per year.

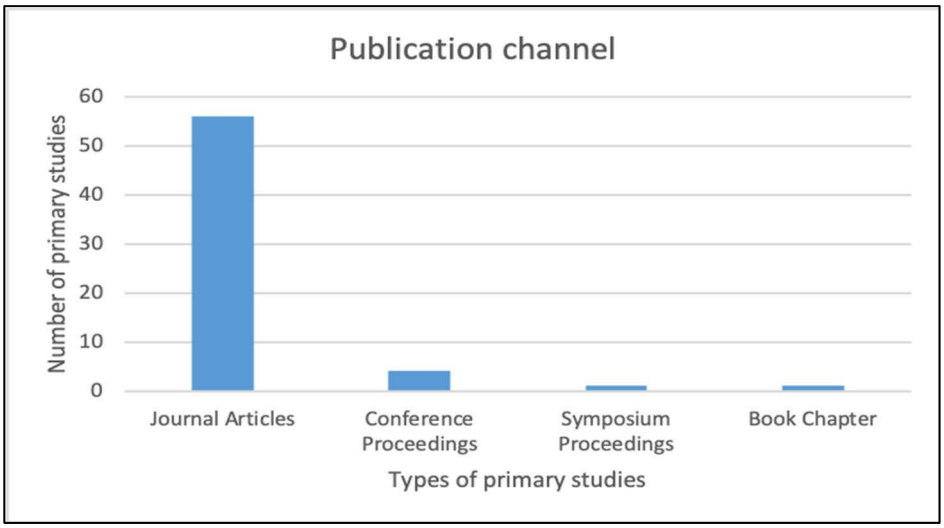

FIGURE 6. Publication channels.

TABLE 8. Top 10 represented publications.

Title	Number of papers
Empirical Software Engineering	8
IET Software	5
Information and Software Technology	5
IEEE Transactions on Software Engineering	5
IEEE Access	3
Sustainability	3
Applied Sciences	3
The Journal of Systems and Software	3
Enterprise Information Systems	2
IEEE Transactions on Reliability	2

Approaches that perform source code mining and analysis on a massive scale can be expensive. A complementary technique has been used to reduce the amount of computation performed by the ultra-large-scale source code mining tasks without compromising the accuracy of the results [19]. These discussions are a rich source of information that can be used to build multiple kinds of support tools for developers. Integrating convolution neural networks with the normalization approach can serve as a support tool to help in rectifying misclassified issue reports [82].

F. RQ6: WHAT ARE THE DEMOGRAPHICS OF THE PRIMARY STUDIES?

To answer this RQ, three aspects of the primary studies were examined: the publication year, the publication type, and the publication that has published the most relevant studies (journals and proceedings).

1) PUBLICATION YEAR

From the period (2016–2021), 60 publications were extracted from the literature, following the study methodology (Section III). Fig. 5 shows the evolution of the publication of AI techniques in the literature. Research activity involving the AI techniques used in SE is progressive and active. From 2016 to 2017, the research activity was linear with few publications. However, in 2018, the research activity involving AI techniques increased significantly, with 13 publications. Interest increased considerably due to the growing demand for AI techniques in SE. Therefore, in 2020, the research activity notably increased, with 37 publications, more than any other year in this analysis. In 2019, a slight decrease in publications was evident, with three papers. Moreover, in 2021, three journal articles were published. It is also important to highlight that no firm and valid conclusion can be made for 2021 because the current research covers only part of that year. Generally, despite the number of publications on AI techniques research changing from year to year, the research activity continues to increase and the research area shows stable growth, particularly during the last six years.

2) PUBLICATION TYPES

In this mapping study, the authors covered 28 different journals, two conference proceedings, one symposium

proceedings, and one book chapter. As Fig. 6 shows, most primary studies were journal articles (56), followed by conference proceedings (4), symposium proceedings (1), and, lastly, book chapters (1).

3) PUBLICATION WITH RELEVANT STUDIES

With respect to the publication venues in which studies of AI techniques in SE were published, Table 8 shows the ten most active journals. Empirical Software Engineering was the top contributor among all the journals, with eight publications. Meanwhile, the IET Software, Information, and Software Technology journal and the IEEE Transactions on Software Engineering journal each provided five articles. The other selected studies published three articles in the following journals: IEEE Access, Sustainability, Applied Sciences, and the Journal of Systems and Software.

G. THREATS TO VALIDITY

We have to discuss the following threats to the validity of the study for the future improvement of mapping studies.

1) CONSTRUCT VALIDITY

The findings that we discuss most likely are valid only for our collection of papers. However, we have attempted for the inclusion of as many relevant papers as possible. To achieve this, we focus on the most relevant digital libraries that might include the application of AI techniques in any of the SE phases. Furthermore, we did not restrict our search to papers mentioning "Artificial Intelligence techniques" explicitly but also included terms closely related to artificial intelligence, without narrowing it to the exact terms for main AI techniques. This enabled us to capture many related AI techniques publications without focusing on the very specific technique of machine or deep learning only. Another threat avoided with regards to the definition of the inclusion/exclusion criteria in the screening activity. We decided to consider the title, abstract, and keywords metadata. We also decided to include most papers we were uncertain about in order to avoid the exclusion of relevant publications due to the lack of investigation. In the first step, all the papers were read and, then, inclusion/exclusion was decided. Finally, to improve construct validity in the future we should also consider papers that might use more diversified terms of the AI techniques.

2) INTERNAL VALIDITY

To alleviate the threat related to the data extraction performed for screening and classification, we decided to discuss some of the papers that contribute to more than one phase of the SE activities. The classification was performed in several settings after analysing the main contributions of each paper. The adopted classification scheme comprises main AI techniques that correspond to the defined research questions.

3) CONCLUSION VALIDITY

Regarding the conclusions, we have discussed and identified only several main AI capabilities that have been applied to support SE activities and phases. For the study's replicability, we provided the complete research method in section 3, which enables replicating every phase of this mapping study.

4) EXTERNAL VALIDITY

The results cannot be generalized to other problems domains.

V. CONCLUSION

This systematic mapping study provides an overview of the contributions and trends involving AI techniques in the SE literature. This study shows that the literature on this topic is heterogeneous, presenting several AI techniques used in SE. A significant proportion of the literature focuses mainly on SE phases, AI techniques, and performance measurement. To conduct this study, the authors used the mostly identified eight SE phases to assess the AI techniques environment. This study found that ML is used the most for automating the process in various SE phases by using a designation algorithm. ML is seen continually evolving and enhancing its efficiency and accuracy. Primarily, ML has been intensively used for automation and improvement activities in the RE phase. On the other hand, ML also enables the automation of various decision-making tasks and significantly assists in SE activities that require prediction. ML is a critical component of predictive analysis. Furthermore, it was noticed that hybrid techniques using ML with other AI techniques further enhanced the value of the performance metrics in their respective evaluation. This combination is effective at handling multidimensional and multi-variety data, which can be achieved in dynamic or uncertain environments. While carrying out this systematic mapping, it became clear that ML techniques have been heavily used compared to other AI techniques. Additionally, it was observed that AI techniques have great potential to be relevantly utilized in many more activities of software development. Several SE phases were identified in which very few AI techniques have been applied. The next research step was to investigate further and then refine the key findings to identify other potential activities within SE phases, such as designing and testing, in which AI techniques could be utilized more in the future.

APPENDIX

This section gives an overview of the research involving AI techniques in SE by briefly summarizing each study.

PS1: Siavvas *et al.* proposed an ML model to evaluate the capacity of technical debt indicators to predict the existence of vulnerabilities in software classes [60].

PS2: Naseer *et al.* employed the cooperative learning environment with the aid of technology to determining the best revolution of the project-oriented academic environment. The proposed approach offers a direct way to the triumph of sustainable education [18].

PS3: Khan *et al.* proposed the CrowdRE-Arg approach to utilize future user forums and other social media platforms to make them more intuitive and user-friendly [28].

PS4: Anh Do *et al.* aimed to evaluate the initial human subject using a feature descriptions framework from three application domains. The results demonstrate the framework's ability to generate creative requirements [55].

PS5: Khan *et al.* set out an automated approach, which automatically identifies the rationale and requirements-related information using machine learning algorithms from the Reddit forum [83].

PS6: Marcen *et al.* presented the TLR-ELtoR approach, which recovers traceability links between the requirements of a software system. This approach may be beneficial for dealing with issues such as tacit knowledge and vocabulary mismatch [75].

PS7: Garcia *et al.* aimed to support expertise location through an ontology that can link information about programmers or any team member with the resources used in a project [84].

PS8: Haider *et al.* proposed using a TFS repository framework for communication and coordination purposes of the GSD, CBR to find their ranking from previous requirements, and then J48 as the classification [29].

PS9: Al-Hroob *et al.* described a semi-automated approach to classifying an intelligent CASE (I-CASE)2. This utilizes the standard NLP and ANN approaches, both AI, to extract actors and actions from the descriptions of use case models' NL requirements [85].

PS10: Li *et al.* proposed a set of useful ML algorithms to discover a wide spectrum of traceability links [31].

PS11: Elias *et al.* showed that the delivery speed prediction performance of software enhancement projects with a search method based on feature construction was statistically better than the performance obtained with multiple line regression when the unadjusted function point and the number of practitioners were used as the independent variables [39].

PS12: Kumar *et al.* introduced prediction models for predicting changes in internal quality attributes using source-code metrics and machine learning algorithms. This would help to improve the software quality and reduce the cost of fixing these bugs. It helps the tester to determine the process of fixing bugs, i.e., changing the design or simple logic of the software [33].

PS13: Watanabe *et al.* presented a detailed approach to reducing the number of false negatives in the classification process [86].

PS14: Saidani *et al.* investigated the GP adaptation prediction rules as a combination of metrics and threshold values, which should correctly predict, as far as possible, the failed builds extracted from a base of real-world examples [50].

PS15: Ding and Xing proposed a new software defect prediction method (PHIForest) based on the improved histogram-based isolation forest, which can mitigate some of the side effects caused by an imbalanced training dataset and enhance prediction performance [49].

PS16: Palomba and Tamburri proposed a project model that can be effectively exploited by practitioners to predict community smells, especially when trained using Random Forest as the classifier [67].

PS17: Ranjan Bal and Kumar proposed a simple and efficient WR-ELM model for predicting the number of software faults [87].

PS18: Kumar *et al.* proposed a model to help the tester to find the process of fixing bugs, i.e., changing the design or the simple logic of the software [33].

PS19: Agrawal *et al.* proposed a DUO that can lead to better (e.g., faster, more accurate, more reliable, more interpretable, and multi-goal) analyses in empirical software engineering studies, in particular those studies that require automated tools to analyze (large quantities of) data [76].

PS20: Nasser *et al.* presented technology-oriented learning, a project-oriented educational environment that can also benefit from ML classifiers for the early prediction of poor performers and lead to better team–instructor synchronization [65].

PS21: Malhotra and Lata introduced a Safe-Level-SMOTE method to handle imbalanced data and improve the performance of ML techniques when developing efficient maintainability prediction models to forecast high maintainability effort classes at the early stages of software development, which is crucial in any software project [42].

PS22: Blincoe *et al.* investigated better support for predicting when high-level requirements will be completed, which would help project managers to make better planning decisions [20].

PS23: Daniel and Slowik presented a predictive model for proper validation to predict future defects in a given project [40].

PS24: Diwaker *et al.* gave an overview stating that fuzzy logic is more compatible for predicting reliability than PSO [51].

PS25: Malhotra and Khanna *et al.* reported extracts of remedial actions from the literature that should be followed by future researchers to ensure an effective experimental setup that would yield practical and reliable results while developing SPMs using SBAs [78].

PS26: Malhotra and Khanna *et al.* provided insights to software practitioners and researchers from an efficient selection of the available techniques [59].

PS27: Choetkiertikul *et al.* developed an accurate model to predict whether an issue would be at risk of being delayed and, if so, the extent of the delay (risk impact), and the likelihood of the risk occurring [21].

PS28: Ryu and Baik introduced effective multi-objective learning techniques in cross-project environments [22].

PS29: Dargan *et al.* reported that requirement quality can be used as a predictive factor for end-system operational performance [70].

PS30: Canedo and Mendes proposed an Automatic Software Requirements classification to help developers to document their projects more effectively, minimize rework, and make the software easier to use and understand [80].

PS31: Rahimi *et al.* introduced an ensemble model that enhances the weighted voting ensemble by using the accuracy per class for each base ensemble to determine the best classification for FR, thus enhancing accuracy and availability [26].

PS32: Li *et al.* proposed a set of useful ML algorithms that can be adopted to discover a wide spectrum of traceability links [31].

PS33: Rodriguez *et al.* reported that interaction design patterns help in complying with the principles of usability and user experience, thus promoting the usage of standards and best practices [27].

PS34: Fujii *et al.* presented the current version QA4A1, which was constructed in a bottom-up, best-effort manner to identify what was missing from the guidelines or the knowledge of research communities [71].

PS35: Agrawal *et al.* recommended combining data mining and optimization using DUO for software analytics as this was possible and useful [76].

PS36: Nayebi *et al.* investigated where abstraction and synthesis are automated steps in the Gandhi Washington method, which condenses the data and later aggregates sequences of items based on their commonality in structure and their effect on a software performance measure [41].

PS37: Theisen and Williams investigated how the Naive Bayes consistently outperformed the other three classifiers in the case study. Combining features from multiple models did not necessarily result in increased accuracy and sometimes resulted in a degradation in performance. While it was thought possible that the models could cover some of the same vulnerabilities, it was unexpected that some features could prove to be distractors or have a negative effect on accuracy [37].

PS38: Ghunaim and Dichter presented the Fuzzy Analytical Hierarchy Process (FAHP) approach, which will increase software developers' confidence in research outcomes and help them to avoid false conclusions and infer reasonable boundaries [73].

PS39: A new defect prediction model framework based on atomic class-association rule discovery was presented by Shao *et al.* It included data pre-processing, rule model building, and performance evaluation. Redundant pruning considers the interaction between features, which improves the prediction performance of the proposed algorithm. The grid-search method is used to control true and false minimum support [23].

PS40: Moustafa *et al.* reported that ensemble classifiers had solved the problem of class-imbalanced datasets by improving their classification accuracy [24].

PS41: Upadhyaya and Rajan introduced a complementary technique that reduces the amount of computation performed by the ultra-large-scale source code mining tasks without compromising the accuracy of the results [19].

PS42: Rahman and Islam proposed that a Decision Tree for small- and medium-sized software and ELM for large-sized software can be a solution that enables software delivery within the scheduled time [69].

PS43: Figalist *et al.* proposed a foster model that shared and reused knowledge, components, tools, and concepts to speed up the instantiation process [43].

PS44: Papamichail and Symeodis maintained that the evaluation methodology targeted multimedia projects that harness information residing in code hosting facilities [57].

PS45: Choetkiertikul *et al.* employed a deep learning-based, fully end-to-end prediction system for estimating story points, removing the users from manually designing the features of the textual description of issues [62].

PS46: Brezocnik *et al.* introduced swarm intelligence methods that are beneficial for solving agile software development tasks [52].

PS47: Pickerill *et al.* proposed PHANTOM, which improves on existing methods with respect to analysis time and hardware requirements but without sacrificing accuracy [74].

PS48: Al-Hawari *et al.* employed an associative classification review mining approach to enable the automatic classification of application reviews into software maintenance tasks [88].

PS49: Xin Xia *et al.* presented integrating CNN with the normalization approach to serve as a support tool that would help in rectifying misclassified issue reports [82].

PS50: Amreen *et al.* proposed a new approach (ALFAA) for correcting identity errors in the SE context and applied it in the OpenStack ecosystem [44].

PS51: Liu *et al.* described how developers get useful information according to their demands from two aspects: security requirements for the whole app and those for the given functionality [45].

PS52: Jiang *et al.* employed crowdsourcing to facilitate attribute construction to improve the task of bug report summarization [34].

PS53: Choetkiertikul *et al.* proposed a novel approach to delivery-related risk prediction in iterative development settings. This approach could predict how much work gets done in the iteration [46].

PS54: Trautsch *et al.* presented SmartSHARK, which can execute different plug-ins (e.g., their vcsSHARK or mecoSHARK) for an arbitrary git-using project [58].

PS55: Hilton and Gethner reported that since many SE organizations do not yet have a large codebase when they most want to be able to predict external quality issues, this generalizability may allow organizations to train models on other projects and then use those models to predict the quality issues that may affect their own [68].

PS56: Diamantopoulos and Symeonidis proposed an effective model for providing useful recommendations to add to a system or improve already identified requirements. Semantic information allows flexibility and traceability when it comes to maintaining and updating the original software requirements [81].

PS57: Catal and Guldan proposed a model that combines the power of five high-performance individual classifiers and uses the majority voting aggregation rule. Since it does not rely on a single classifier and evaluates the result of each classifier, its generalization capability is high [47].

PS58: Iqbal *et al.* considered that inverse requirements were error-free and easily changeable according to customer requirements [36].

PS59: Bettaieb *et al.* described how classification models derived from historical data largely align with expert reasoning about the applicability of security controls [35].

PS60: Chandra *et al.* investigated how the time invested in testing software during the software cycle can be reduced to datasets that incur changes and errors. This process can be performed efficiently and effectively [32].

REFERENCES

[1] H. K. Dam, "Artificial intelligence for software engineering," *XRDS, Crossroads, ACM Mag. Students*, vol. 25, no. 3, pp. 34–37, Apr. 2019, doi: 10.1145/3313117.

[2] D. P. Wangoo, "Artificial intelligence techniques in software engineering for automated software reuse and design," in *Proc. 4th Int. Conf. Comput. Commun. Autom. (ICCCA)*, Dec. 2018, pp. 1–4, doi: 10.1109/CCAA.2018.8777584.

[3] A. Ahmad, C. Feng, M. Khan, A. Khan, A. Ullah, S. Nazir, and A. Tahir, "A systematic literature review on using machine learning algorithms for software requirements identification on stack overflow," *Secur. Commun. Netw.*, vol. 2020, pp. 1–19, Jul. 2020, doi: 10.1155/2020/8830683.

[4] H. Alsolai and M. Roper, "A systematic literature review of machine learning techniques for software maintainability prediction," *Inf. Softw. Technol.*, vol. 119, Mar. 2020, Art. no. 106214, doi: 10.1016/j.infsof.2019.106214.

[5] N. Li, M. Shepperd, and Y. Guo, "A systematic review of unsupervised learning techniques for software defect prediction," *Inf. Softw. Technol.*, vol. 122, Jun. 2020, Art. no. 106287, doi: 10.1016/j.infsof.2020.106287.

[6] R. Malhotra, "A systematic review of machine learning techniques for software fault prediction," *Appl. Soft Comput.*, vol. 27, pp. 504–518, Feb. 2015, doi: 10.1016/j.asoc.2014.11.023.

[7] S. K. Pandey, R. B. Mishra, and A. K. Tripathi, "Machine learning based methods for software fault prediction: A survey," *Expert Syst. Appl.*, vol. 172, Jun. 2021, Art. no. 114595, doi: 10.1016/j.eswa.2021.114595.

[8] M. I. Azeem, F. Palomba, L. Shi, and Q. Wang, "Machine learning techniques for code smell detection: A systematic literature review and meta-analysis," *Inf. Softw. Technol.*, vol. 108, pp. 115–138, Apr. 2019, doi: 10.1016/j.infsof.2018.12.009.

[9] M. Binkhonain and L. Zhao, "A review of machine learning algorithms for identification and classification of non-functional requirements," *Expert Syst. Appl., X*, vol. 1, Apr. 2019, Art. no. 100001, doi: 10.1016/j.eswax.2019.100001.

[10] 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, doi: 10.1109/TR.2019.2892517.

[11] D. El-Masri, F. Petrillo, Y.-G. Guéhéneuc, A. Hamou-Lhadj, and A. Bouziane, "A systematic literature review on automated log abstraction techniques," *Inf. Softw. Technol.*, vol. 122, Jun. 2020, Art. no. 106276, doi: 10.1016/j.infsof.2020.106276.

[12] N. Genc-Nayebi and A. Abran, "A systematic literature review: Opinion mining studies from mobile app store user reviews," *J. Syst. Softw.*, vol. 125, pp. 207–219, Mar. 2017, doi: 10.1016/j.jss.2016.11.027.

[13] H. Villamizar, T. Escovedo, and M. Kalinowski, "Requirements engineering for machine learning: A systematic mapping study," in *Proc. 47th Euromicro Conf. Softw. Eng. Adv. Appl. (SEAA)*, Sep. 2021, pp. 29–36, doi: 10.1109/SEAA53835.2021.00013.

[14] A. Zakari, S. P. Lee, K. A. Alam, and R. Ahmad, "Software fault localisation: A systematic mapping study," *IET Softw.*, vol. 13, no. 1, pp. 60–74, Feb. 2018, doi: 10.1049/iet-sen.2018.5137.

[15] K. Petersen, R. Feldt, S. Mujtaba, and M. Mattsson, "Systematic mapping studies in software engineering," in *Proc. Electron. Workshops Comput.*, Jun. 2008, pp. 1–10, doi: 10.14236/ewic/ease2008.8.

[16] B. Kitchenham, O. P. Brereton, D. Budgen, M. Turner, J. Bailey, and S. Linkman, "Systematic literature reviews in software engineering—A systematic literature review," *Inf. Softw. Technol.*, vol. 51, no. 1, pp. 7–15, Jan. 2009, doi: 10.1016/j.infsof.2008.09.009.

[17] Z. Jin, "Requirements engineering methodologies," in *Environment Modeling-Based Requirements Engineering for Software Intensive Systems*. Amsterdam, The Netherlands: Elsevier, 2018, pp. 13–27, doi: 10.1016/b978-0-12-801954-2.00002-9.

[18] 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, doi: 10.3390/su12218986.

[19] G. Upadhyaya and H. Rajan, "On accelerating source code analysis at massive scale," *IEEE Trans. Softw. Eng.*, vol. 44, no. 7, pp. 669–688, Jul. 2018, doi: 10.1109/TSE.2018.2828848.

[20] K. Blincoe, A. Dehghan, A.-D. Salaou, A. Neal, J. Linaker, and D. Damian, "High-level software requirements and iteration changes: A predictive model," *Empirical Softw. Eng.*, vol. 24, no. 3, pp. 1610–1648, Jun. 2019, doi: 10.1007/s10664-018-9656-z.

[21] M. Choetkiertikul, H. K. Dam, T. Tran, and A. Ghose, "Predicting the delay of issues with due dates in software projects," *Empirical Softw. Eng.*, vol. 22, no. 3, pp. 1223–1263, Jun. 2017, doi: 10.1007/s10664-016-9496-7.

[22] D. Ryu and J. Baik, "Effective multi-objective naïve Bayes learning for cross-project defect prediction," *Appl. Soft Comput.*, vol. 49, pp. 1062–1077, Dec. 2016, doi: 10.1016/j.asoc.2016.04.009.

[23] Y. Shao, B. Liu, S. Wang, and G. Li, "A novel software defect prediction based on atomic class-association rule mining," *Expert Syst. Appl.*, vol. 114, pp. 237–254, Dec. 2018, doi: 10.1016/j.eswa.2018.07.042.

[24] S. Moustafa, M. Y. ElNainay, N. E. Makky, and M. S. Abougabal, "Software bug prediction using weighted majority voting techniques," *Alexandria Eng. J.*, vol. 57, no. 4, pp. 2763–2774, Dec. 2018, doi: 10.1016/j.aej.2018.01.003.

[25] E. D. Canedo and B. C. Mendes, "Software requirements classification using machine learning algorithms," *Entropy*, vol. 22, no. 9, p. 1057, Sep. 2020, doi: 10.3390/E22091057.

[26] N. Rahimi, F. Eassa, and L. Elrefaei, "An ensemble machine learning technique for functional requirement classification," *Symmetry*, vol. 12, no. 10, pp. 1–26, Oct. 2020, doi: 10.3390/sym12101601.

[27] V. Silva-Rodríguez, S. E. Nava-Muñoz, L. A. Castro, F. E. Martínez-Pérez, H. G. Pérez-González, and F. Torres-Reyes, "Classifying design-level requirements using machine learning for a recommender of interaction design patterns," *IET Softw.*, vol. 14, no. 5, pp. 544–552, Oct. 2020, doi: 10.1049/iet-sen.2019.0291.

[28] J. A. Khan, L. Liu, L. Wen, and R. Ali, "Conceptualising, extracting and analysing requirements arguments in users' forums: The CrowdRE-Arg framework," *J. Softw., Evol. Process.*, vol. 32, no. 12, pp. 1–37, Dec. 2020, doi: 10.1002/smr.2309.

[29] W. Haider, M. Jawad, Y. Hafeez, F. B. Ahmad, S. Ali, and M. N. Rafi, "Improving requirement prioritization and traceability using artificial intelligence technique for global software development," in *Proc. 22nd Int. Multitopic Conf.*, 2019, pp. 1–8.

[30] A. Al-Hroob, A. T. Imam, and R. Al-Heisa, "The use of artificial neural networks for extracting actions and actors from requirements document," *Inf. Softw. Technol.*, vol. 101, pp. 1–15, Sep. 2018, doi: 10.1016/j.infsof.2018.04.010.

[31] T. Li, S. Wang, D. Lillis, and Z. Yang, "Combining machine learning and logical reasoning to improve requirements traceability recovery," *Appl. Sci.*, vol. 10, no. 20, pp. 1–23, Oct. 2020, doi: 10.3390/app10207253.

[32] 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, doi: 10.1109/ICICCS.2016.7542340.

[33] L. Kumar, S. Tummalapalli, and L. B. Murthy, "An empirical framework to investigate the impact of bug fixing on internal quality attributes," *Arabian J. Sci. Eng.*, vol. 46, no. 4, pp. 3189–3211, Apr. 2021, doi: 10.1007/s13369-020-05095-0.

[34] H. Jiang, X. Li, Z. Ren, J. Xuan, and Z. Jin, "Toward better summarizing bug reports with crowdsourcing elicited attributes," *IEEE Trans. Rel.*, vol. 68, no. 1, pp. 2–22, Mar. 2019, doi: 10.1109/TR.2018.2873427.

[35] S. Bettaieb, S. Y. Shin, M. Sabetzadeh, L. C. Briand, M. Garceau, and A. Meyers, "Using machine learning to assist with the selection of security controls during security assessment," *Empirical Softw. Eng.*, vol. 25, no. 4, pp. 2550–2582, Jul. 2020, doi: 10.1007/s10664-020-09814-x.

[36] N. Iqbal, J. Sang, J. Chen, and X. Xia, "Measuring software maintainability with naïve Bayes classifier," *Entropy*, vol. 23, no. 2, pp. 1–27, Feb. 2021, doi: 10.3390/e23020136.

[37] C. Theisen and L. Williams, "Better together: Comparing vulnerability prediction models," *Inf. Softw. Technol.*, vol. 119, Mar. 2020, Art. no. 106204, doi: 10.1016/j.infsof.2019.106204.

[38] J. A. Khan, L. Liu, and L. Wen, "Requirements knowledge acquisition from online user forums," *IET Softw.*, vol. 14, no. 3, pp. 242–253, Jun. 2020, doi: 10.1049/iet-sen.2019.0262.

[39] E. Ventura-Molina, C. López-Martín, I. López-Yáñez, and C. Yáñez-Márquez, "A novel data analytics method for predicting the delivery speed of software enhancement projects," *Mathematics*, vol. 8, no. 11, pp. 1–22, Nov. 2020, doi: 10.3390/math8112002.

[40] D. Czyczyn-Egird and A. Slowik, "Defect prediction in software using predictive models based on historical data," in *Proc. Adv. Intell. Syst. Comput.*, vol. 801, 2019, pp. 96–103, doi: 10.1007/978-3-319-99608-0_11.

[41] M. Nayebi, G. Ruhe, and T. Zimmermann, "Mining treatment-outcome constructs from sequential software engineering data," *IEEE Trans. Softw. Eng.*, vol. 47, no. 2, pp. 393–411, Feb. 2021, doi: 10.1109/TSE.2019.2892956.

[42] R. Malhotra and K. Lata, "An empirical study on predictability of software maintainability using imbalanced data," *Softw. Quality J.*, vol. 28, no. 4, pp. 1581–1614, Dec. 2020, doi: 10.1007/s11219-020-09525-y.

[43] I. Figalist, C. Elsner, J. Bosch, and H. H. Olsson, "Fast and curious: A model for building efficient monitoring- and decision-making frameworks based on quantitative data," *Inf. Softw. Technol.*, vol. 132, Apr. 2021, Art. no. 106458, doi: 10.1016/j.infsof.2020.106458.

[44] S. Amreen, A. Mockus, R. Zaretzki, C. Bogart, and Y. Zhang, "ALFAA: Active learning fingerprint based anti-aliasing for correcting developer identity errors in version control systems," *Empirical Softw. Eng.*, vol. 25, no. 2, pp. 1136–1167, Mar. 2020, doi: 10.1007/s10664-019-09786-7.

[45] Y. Liu, L. Liu, H. Liu, S. Gao, and G. Song, "Recommending security requirements for the development of Android applications based on sensitive APIs," *IEEE Access*, vol. 8, pp. 101591–101606, 2020, doi: 10.1109/ACCESS.2020.2997335.

[46] M. Choetkiertikul, H. K. Dam, T. Tran, A. Ghose, and J. Grundy, "Predicting delivery capability in iterative software development," *IEEE Trans. Softw. Eng.*, vol. 44, no. 6, pp. 551–573, Jun. 2018, doi: 10.1109/TSE.2017.2693989.

[47] C. Catal and S. Guldan, "Product review management software based on multiple classifiers," *IET Softw.*, vol. 11, no. 3, pp. 89–92, Jun. 2017, doi: 10.1049/iet-sen.2016.0137.

[48] E. Algorithms, "Heuristic algorithms," *Ann. Discrete Math.*, vol. 53, no. C, pp. 221–242, 1992, doi: 10.1016/S0167-5060(08)70205-X.

[49] Z. Ding and L. Xing, "Improved software defect prediction using pruned histogram-based isolation forest," *Rel. Eng. Syst. Saf.*, vol. 204, Dec. 2020, Art. no. 107170, doi: 10.1016/j.ress.2020.107170.

[50] I. Saidani, A. Ouni, M. Chouchen, and M. W. Mkaouer, "Predicting continuous integration build failures using evolutionary search," *Inf. Softw. Technol.*, vol. 128, Dec. 2020, Art. no. 106392, doi: 10.1016/j.infsof.2020.106392.

[51] C. Diwaker, P. Tomar, A. Solanki, A. Nayyar, N. Z. Jhanjhi, A. Abdullah, and M. Supramaniam, "A new model for predicting component-based software reliability using soft computing," *IEEE Access*, vol. 7, pp. 147191–147203, 2019, doi: 10.1109/ACCESS.2019.2946862.

[52] L. Brezoínik, I. Fister, and V. Podgorelec, *Solving Agile Software Development Problems With Swarm Intelligence Algorithms* (Lecture Notes in Networks and Systems), vol. 76. Cham, Switzerland: Springer, 2020, pp. 298–309.

[53] A. M. T. Torres, "Machine learning approaches for error correction of hydraulic simulation models for canal flow schemes," Utah Water Res. Lab., Utah State Univ., Logan, UT, USA, Tech. Rep., 2008.

[54] A. Chahal and P. Gulia, "Machine learning and deep learning," *Int. J. Innov. Technol. Explor. Eng.*, vol. 8, no. 12, pp. 4910–4914, 2019, doi: 10.35940/ijitee.L3550.1081219.

[55] Q. A. Do, T. Bhowmik, and G. L. Bradshaw, "Capturing creative requirements via requirements reuse: A machine learning-based approach," *J. Syst. Softw.*, vol. 170, Dec. 2020, Art. no. 110730, doi: 10.1016/j.jss.2020.110730.

[56] 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, doi: 10.1016/j.jss.2020.110519.

[57] M. D. Papamichail and A. L. Symeonidis, "Data-driven analytics towards software sustainability: The case of open-source multimedia tools on cultural storytelling," *Sustainability*, vol. 13, no. 3, pp. 1–16, Feb. 2021, doi: 10.3390/su13031079.

[58] F. Trautsch, S. Herbold, P. Makedonski, and J. Grabowski, "Addressing problems with replicability and validity of repository mining studies through a smart data platform," *Empirical Softw. Eng.*, vol. 23, no. 2, pp. 1036–1083, Apr. 2018, doi: 10.1007/s10664-017-9537-x.

[59] R. Malhotra and M. Khanna, "An exploratory study for software change prediction in object-oriented systems using hybridized techniques," *Automated Softw. Eng.*, vol. 24, no. 3, pp. 673–717, Sep. 2017, doi: 10.1007/s10515-016-0203-0.

[60] M. Siavvas, D. Tsoukalas, M. Jankovic, D. Kehagias, and D. Tzovaras, "Technical debt as an indicator of software security risk: A machine learning approach for software development enterprises," *Enterprise Inf. Syst.*, vol. 16, no. 5, pp. 1–44, May 2022, doi: 10.1080/17517575.2020.1824017.

[61] P. R. Bal and S. Kumar, "WR-ELM: Weighted regularization extreme learning machine for imbalance learning in software fault prediction," *IEEE Trans. Rel.*, vol. 69, no. 4, pp. 1355–1375, Dec. 2020, doi: 10.1109/TR.2020.2996261.

[62] M. Choetkiertikul, H. K. Dam, T. Tran, T. Pham, A. Ghose, and T. Menzies, "A deep learning model for estimating story points," *IEEE Trans. Softw. Eng.*, vol. 45, no. 7, pp. 637–656, Jul. 2019, doi: 10.1109/TSE.2018.2792473.

[63] M. Siavvas, D. Tsoukalas, M. Jankovic, D. Kehagias, and D. Tzovaras, "Technical debt as an indicator of software security risk: A machine learning approach for software development enterprises," *Enterprise Inf. Syst.*, vol. 16, no. 5, pp. 1–43, May 2022, doi: 10.1080/17517575.2020.1824017.

[64] I. Saidani, A. Ouni, M. Chouchen, and M. W. Mkaouer, "Predicting continuous integration build failures using evolutionary search," *Inf. Softw. Technol.*, vol. 128, Dec. 2020, Art. no. 106392, doi: 10.1016/j.infsof.2020.106392.

[65] M. Naseer, W. Zhang, and W. Zhu, "Early prediction of a team performance in the initial assessment phases of a software project for sustainable software engineering education," *Sustainability*, vol. 12, no. 11, p. 4663, Jun. 2020, doi: 10.3390/su12114663.

[66] 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, doi: 10.3390/su12218986.

[67] F. Palomba and D. A. Tamburri, "Predicting the emergence of community smells using socio-technical metrics: A machine-learning approach," *J. Syst. Softw.*, vol. 171, Jan. 2021, Art. no. 110847, doi: 10.1016/j.jss.2020.110847.

[68] R. Hilton and E. Gethner, "Predicting code hotspots in open-source software from object-oriented metrics using machine learning," *Int. J. Softw. Eng. Knowl. Eng.*, vol. 28, no. 3, pp. 311–331, Mar. 2018, doi: 10.1142/S0218194018500110.

[69] M. T. Rahman and M. M. Islam, "A comparison of machine learning algorithms to estimate effort in varying sized software," in *Proc. IEEE Region Symp.*, Jun. 2019, pp. 137–142.

[70] J. L. Dargan, J. S. Wasek, and E. Campos-Nanez, "Systems performance prediction using requirements quality attributes classification," *Requirements Eng.*, vol. 21, no. 4, pp. 553–572, Nov. 2016, doi: 10.1007/s00766-015-0232-4.

[71] G. Fujii, K. Hamada, F. Ishikawa, S. Masuda, M. Matsuya, T. Myojin, Y. Nishi, H. Ogawa, T. Toku, S. Tokumoto, K. Tsuchiya, and Y. Ujita, "Guidelines for quality assurance of machine learning-based artificial intelligence," *Int. J. Softw. Eng. Knowl. Eng.*, vol. 30, nos. 11–12, pp. 1589–1606, Nov. 2020, doi: 10.1142/S0218194020400227.

[72] J. Ali Khan, L. Liu, L. Wen, and R. Ali, "Conceptualising, extracting and analysing requirements arguments in users' forums: The CrowdRE-Arg framework," *J. Softw., Evol. Process.*, vol. 32, no. 12, pp. 1–37, Dec. 2020, doi: 10.1002/smr.2309.

[73] H. Ghunaim and J. Dichter, "Applying the FAHP to improve the performance evaluation reliability of software defect classifiers," *IEEE Access*, vol. 7, pp. 62794–62804, 2019, doi: 10.1109/ACCESS.2019.2915964.

[74] P. Pickerill, H. J. Jungen, M. Ochodek, M. Maćkowiak, and M. Staron, "PHANTOM: Curating GitHub for engineered software projects using time-series clustering," *Empirical Softw. Eng.*, vol. 25, no. 4, pp. 2897–2929, Jul. 2020, doi: 10.1007/s10664-020-09825-8.
[75] 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, doi: 10.1016/j.jss.2020.110519.
[76] A. Agrawal, T. Menzies, L. L. Minku, M. Wagner, and Z. Yu, "Better software analytics via 'DUO': Data mining algorithms using/used-by optimizers," *Empirical Softw. Eng.*, vol. 25, no. 3, pp. 2099–2136, May 2020, doi: 10.1007/s10664-020-09808-9.
[77] W. M. Watanabe, K. R. Felizardo, A. Candido, É. F. de Souza, J. E. D. C. Neto, and N. L. Vijaykumar, "Reducing efforts of software engineering systematic literature reviews updates using text classification," *Inf. Softw. Technol.*, vol. 128, Dec. 2020, Art. no. 106395, doi: 10.1016/j.infsof.2020.106395.
[78] R. Malhotra and M. Khanna, "Threats to validity in search-based predictive modelling for software engineering," *IET Softw.*, vol. 12, no. 4, pp. 293–305, Aug. 2018, doi: 10.1049/iet-sen.2018.5143.
[79] W. Haider, Y. Hafeez, S. Ali, M. Jawad, F. B. Ahmad, and M. N. Rafi, "Improving requirement prioritization and traceability using artificial intelligence technique for global software development," in *Proc. 22nd Int. Multitopic Conf. (INMIC)*, Nov. 2019, pp. 1–8, doi: 10.1109/INMIC48123.2019.9022775.
[80] E. D. Canedo and B. C. Mendes, "Software requirements classification using machine learning algorithms," *Entropy*, vol. 22, no. 9, p. 1057, Sep. 2020, doi: 10.3390/E22091057.
[81] T. Diamantopoulos and A. Symeonidis, "Enhancing requirements reusability through semantic modeling and data mining techniques," *Enterprise Inf. Syst.*, vol. 12, nos. 8–9, pp. 960–981, Oct. 2018, doi: 10.1080/17517575.2017.1416177.
[82] Q. Huang, X. Xia, D. Lo, and G. C. Murphy, "Automating intention mining," *IEEE Trans. Softw. Eng.*, vol. 46, no. 10, pp. 1098–1119, Oct. 2020, doi: 10.1109/TSE.2018.2876340.
[83] J. Ali Khan, L. Liu, and L. Wen, "Requirements knowledge acquisition from online user forums," *IET Softw.*, vol. 14, no. 3, pp. 242–253, Jun. 2020, doi: 10.1049/iet-sen.2019.0262.
[84] J. R. Martínez-García, F.-E. Castillo-Barrera, R. R. Palacio, G. Borrego, and J. C. Cuevas-Tello, "Ontology for knowledge condensation to support expertise location in the code phase during software development process," *IET Softw.*, vol. 14, no. 3, pp. 234–241, Jun. 2020, doi: 10.1049/iet-sen.2019.0272.
[85] A. Al-Hroob, A. T. Imam, and R. Al-Heisa, "The use of artificial neural networks for extracting actions and actors from requirements document," *Inf. Softw. Technol.*, vol. 101, pp. 1–15, Sep. 2018, doi: 10.1016/j.infsof.2018.04.010.
[86] W. M. Watanabe, K. R. Felizardo, A. Candido, É. F. de Souza, J. E. D. C. Neto, and N. L. Vijaykumar, "Reducing efforts of software engineering systematic literature reviews updates using text classification," *Inf. Softw. Technol.*, vol. 128, Dec. 2020, Art. no. 106395, doi: 10.1016/j.infsof.2020.106395.
[87] P. R. Bal and S. Kumar, "WR-ELM: Weighted regularization extreme learning machine for imbalance learning in software fault prediction," *IEEE Trans. Rel.*, vol. 69, no. 4, pp. 1355–1375, Dec. 2020, doi: 10.1109/TR.2020.2996261.
[88] A. Al-Hawari, H. Najadat, and R. Shatnawi, "Classification of application reviews into software maintenance tasks using data mining techniques," *Softw. Qual. J.*, vol. 29, no. 3, pp. 667–703, Sep. 2021, doi: 10.1007/s11219-020-09529-8.

HAZRINA SOFIAN (Senior Member, IEEE) received the Ph.D. degree in intelligent computing from University Putra Malaysia. She is currently a Senior Lecturer with the Department of Software Engineering, Faculty of Computer Science and Information Technology, University of Malaya. She has six years of working experience in the software development industry as a software development project manager, a senior business analyst, and a software analyst. She is actively involved in various research activities in the area of requirements engineering, semantic web/ontology, software automation and natural language processing, and application of artificial intelligence in software engineering. She is also involved in Active Learning and Design Thinking in Engineering Education (ALIEN) Project under the Erasmus Plus. Her research interests include software sustainability and automation in the domain of health, finance, and education.

NUR ARZILAWATI MD YUNUS received the Ph.D. degree in computer networks from Universiti Putra Malaysia, Selangor, Malaysia, in 2022. She is currently working as a Lecturer at the University of Cyberjaya. Her research interests include shuffle-exchange networks, multistage interconnection networks, optical switching, parallel computing, and routing algorithm. She is a member of the International Association of Computer Science and Information Technology Press (IACSIT).

RODINA AHMAD received the bachelor's and master's degrees from USA and the Ph.D. degree in information systems management from the National University of Malaysia, in 2006. In 1991, she started teaching at the Department of Computer Science, University of Technology Malaysia. She moved to the University of Malaya to continue her teaching career, in 1993. She is currently working at the Department of Software Engineering, University of Malaya. She has been very actively involved in various research activities and publications in the area of requirements engineering, e-learning, software process improvement, application of AI in software development, and education. She has published more than 80 publication items in journals and proceedings. She has also presented in various international conferences. Moreover, she has been enthusiastically keen in teaching and supervising handful of master's and Ph.D. students and developing an IT-based approach to help kids with autism. She pursuing her interest in book writing. She has also been actively involved as a journal associate editor and a reviewer of various international journals and conferences.