IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 11 October 2024, accepted 24 October 2024, date of publication 1 November 2024, date of current version 26 November 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3488904

TOPICAL REVIEW

A Decade of Progress: A Systematic Literature Review on the Integration of AI in Software Engineering Phases and Activities (2013-2023)

USMAN KHAN DURRANI[1], (Member, IEEE), MUSTAFA AKPINAR[1,2], (Member, IEEE), MUHAMMED FATIH ADAK[3], (Member, IEEE), ABDULLAH TALHA KABAKUS[4], MUHAMMED MARUF ÖZTÜRK[5], AND MOHAMMED SALEH[1], (Member, IEEE)

[1]Department of Computer and Information Sciences, Higher Colleges of Technology, Dubai, United Arab Emirates
[2]Department of Software Engineering, Sakarya University, 54050 Sakarya, Türkiye
[3]Department of Computer Engineering, Sakarya University, 54050 Sakarya, Türkiye
[4]Department of Computer Engineering, Duzce University, 81620 Düzce, Türkiye
[5]Department of Computer Engineering, Suleyman Demirel University, 32260 Isparta, Türkiye

Corresponding author: Usman Khan Durrani (dr.udurrani@gmail.com)

ABSTRACT The synergy between software engineering (SE) and artificial intelligence (AI) catalyzes software development, as numerous recent studies illustrate an intensified intersection between these domains. This systematic literature review examines the integration of AI techniques or methodologies across SE phases and related activities spanning from 2013 to 2023, resulting in the selection of 110 research papers. Investigating the profound influence of AI techniques, including machine learning, deep learning, natural language processing, optimization algorithms, and expert systems, across various SE phases—such as planning, requirement engineering, design, development, testing, deployment, and maintenance—is the focal point of this study. Notably, the extensive adoption of machine learning and deep learning algorithms in the development and testing phases has enhanced software quality through defect prediction, code recommendation, and vulnerability detection initiatives. Furthermore, natural language processing's role in automating requirements classification and sentiment analysis has streamlined SE practices. Optimization algorithms have also demonstrated efficacy in refining SE activities such as feature location and software repair action predictions, augmenting precision and efficiency in maintenance endeavors. Prospective research emphasizes the imperative of interpretable AI models and the exploration of novel AI paradigms, including explainable AI and reinforcement learning, to promote ethical and efficient software development practices. This paper fills the gap identified in AI techniques dedicated to improving SE phases. The review concludes that AI in SE is revolutionizing the discipline, enhancing software quality, efficiency, and innovation, with ongoing efforts targeting the mitigation of identified limitations and the augmentation of AI capabilities for intelligent and dependable SE.

INDEX TERMS AI, artificial intelligence, deep learning, expert systems, integration, machine learning, natural language processing, optimization algorithms, planning, requirement engineering, software deployment, software development, software engineering, software maintenance, software testing, systematic literature review.

I. INTRODUCTION

Software engineering (SE) has evolved significantly since its inception in the late 1960s, transitioning from ad hoc programming practices to a disciplined approach. This

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

evolution has been driven by the increasing complexity of software systems and the demand for higher quality, reliability, and maintainability. Traditional methodologies, while foundational, often struggle to keep pace with the dynamic requirements and rapid development cycles of modern software projects [1].

The incorporation of Artificial Intelligence (AI) into SE has brought about a significant period of change, transforming the software development cycle with remarkable progress and effectiveness in recent years [2], [3], [4]. This merging of AI with SE signifies a combination of computational ability and inventive issue resolution, altering conventional methods and driving the domain towards advancement and distinction [5], [6]. Across various stages, from initial planning and requirement collection to deployment and maintenance, AI has been instrumental in enhancing efficiency, improving resource management, and tackling intricate issues inherent in SE operations [7], [8].

Initially, merging AI's precise algorithms with the creative problem-solving approach of SE was seen to boost progress and effectiveness [9], [10]. By leveraging AI's capabilities in data analysis, recognizing patterns, and making autonomous decisions, software engineers have gained the ability to address complex design issues, automate routine tasks, and improve overall software quality. This paper's systematic review explores how AI integration transformed SE from 2013 to 2023. It follows the classification of AI applications in SE domain into object-related, function-related, and process-related outlined by [11] to offer structured insight into AI's diverse contributions to SE. We adopt process-related classification for our study, focusing on the various SE phases and associated activities and the influence of AI technologies on enhancing efficiency and accuracy. Concretely, software engineering studies generally include classification and clustering to extract models from a specific corpus. In this respect, we intend to achieve meaningful findings that may shed light on future studies to drive the focus of machine learning in this field.

Unlike the object-related classification, which emphasizes AI outputs like autonomous systems, or the function-related classification that highlights specific technologies such as machine learning or natural language processing, the process-related perspective links AI to distinct SE phases, including development, testing, and maintenance. This classification allows for a systematic and comprehensive evaluation of AI's role at every SE phase, from enhancing project planning activities to automating bug identification during the testing phase. By methodically assessing AI's impact throughout the entire SE life cycle, the process-related perspective provides the necessary depth and scope to understand AI's incorporation into SE phases and associated activities, thus reinforcing its position as the most suitable framework for this analysis.

AI has significantly impacted different SE phases, introduced novel solutions, and enhanced efficiency [12], [13]. Sophisticated machine learning (ML) models have transformed software effort estimation (SEE) in the planning phase, facilitating more precise forecasts and improved project planning and resource management [14], [15]. AI-powered tools have simplified the design phase, automating activities such as architectural design and user interface development and optimizing resource allocation. This provides software engineers with tools that augment design accuracy and minimize manual workload.

In software development, AI-powered systems have significantly accelerated coding processes by providing context-aware suggestions and automating routine coding activities, leading to reduced development time, improved coding standards, and minimized errors. The testing phase has witnessed a profound transformation with AI techniques or methods enhancing test case generation and optimization processes, resulting in more comprehensive testing suites that offer better coverage and efficiency and higher quality [16] software engineering [17], [18]. During deployment and maintenance, AI tools have emerged as pivotal assets in bug prediction and vulnerability detection [19], automating maintenance tasks and ensuring software longevity and quality through proactive issue identification and resolution.

Drawing from the groundwork established through the transformative fusion of AI within SE, it becomes apparent that the journey is ongoing. Despite the notable progress documented, this review of existing literature uncovers critical areas that require further examination and innovative approaches. The urgent demand for specialized datasets within specific domains, the complexities surrounding model intricacy and interpretability, and the perpetual evolution of AI and SE domains suggest unexplored terrain ripe with opportunities. As we delve into emerging AI paradigms such as Deep Learning (DL) and elucidate the concept of explainable AI, novel prospects for automating and augmenting SE tasks emerge, offering potential solutions to increasingly complex challenges. Integrating AI into software engineering not only drives technical progress but also highlights the need to thoughtfully address the social and technical challenges that come with it. Ethical, privacy, and societal concerns emphasize the need for a comprehensive approach, ensuring that the amalgamation of AI and SE progresses in a responsible and socially beneficial manner.

Synthesizing the significant progress with the outlined future directions, the confluence of AI and SE is a dynamic and evolving landscape. The path ahead is rich with pioneering research and development opportunities, inviting a collaborative effort to explore and harness AI's full potential in reshaping the SE landscape.

This study advances the field of SE and AI by conducting a systematic literature review (SLR) from 2013 to 2023. It examines the impact of AI techniques—such as machine learning, deep learning, natural language processing, optimization algorithms, and expert systems—on various SE

phases. It highlights how these technologies enhance the efficiency and accuracy of SE phases.

The subsequent sections of this paper will discuss: Section II explores previous systematic studies on AI integration in SE; Section III details the study's approach and framework; Section IV presents key findings and insights from the SLR, and finally, Section V. summarizes the study's contributions, addressing limitations, and outlining future research directions in the dynamic field of AI and SE integration.

II. RELATED WORKS

The integration of AI into SE may initially appear paradoxical, given AI's focus on automating routine operations and the creative, knowledge-intensive nature of SE. However, previous SLR studies revealed that AI can effectively support creative processes in SE by employing self-optimizing algorithms for organization and optimization tasks
[11], [20], [21].

Reference [11] outlines AI methodologies in SE, accentuating the growing significance of AI in refining software development processes, boosting productivity, and enhancing project success rates. It spotlights ML, heuristic algorithms (HA), and data-driven techniques (DD), emphasizing the importance of evaluation metrics such as precision, recall, F1-score, and accuracy for gauging AI model efficacy. Reference [20] presents a wider, systematic overview of AI methodologies utilized across all SE stages. This work categorizes AI methodologies like ML, DL, and optimization algorithms, pinpointing deficiencies within the literature and advocating for a more cohesive approach to AI in SE. It underscores the necessity for research that encompasses all SE phases and highlights underrepresented domains where AI's complete capabilities are yet to be tapped. The paper stresses that although numerous AI methodologies are employed for specific SE activities such as code generation or testing, the integration throughout the entire software development lifecycle is still lacking, thus creating avenues for future exploration.

Reference [21] delivers an extensive examination of the interplay between AI and SE, offering a historical perspective on the integration of AI and SE from 1984 to 2019. It discusses the reciprocal advantages of this integration, including swift product development, automated debugging, intelligent assistants, and code generation. It classifies the application of AI in SE, SE within AI, and the collaboration between the two.

Another SLR by [22] conducted an automated search for SLRs published between 2004 and 2008. Their findings reveal diverse research topics but limited coverage of core SE areas, emphasizing the need for enhanced quality assessment and practitioner-oriented guidelines. Reference [23] introduces the AI in SE Application Levels (AI-SEAL) taxonomy to categorize AI applications in software systems. This taxonomy classifies applications based on the point of application, AI technology type, and automation level, aiming to aid in understanding risks and guiding strategic decisions on AI integration. AI integration study by [24] examined AI's role in coding, testing, planning, and project estimation. This research highlights how AI techniques can streamline processes, reduce costs, and elevate software quality, paving the way for transformative advancements in SE. An exploratory study [25] implemented AI techniques like machine learning, deep learning, natural language processing, genetic algorithms, and time series analysis in SE for software defect prediction to enhance software quality and efficiency in development, testing, and maintenance.

Reference [1] thoroughly examines AI techniques in SE aligned with IEEE 12207 standard processes, emphasizing the complexity of software systems and AI's role in enhancing quality and time-to-market efficiency. It addresses the disconnect between AI research and practical application, focusing on requirements analysis, architecture design, coding, and testing. It also explores AI applications like natural language processing, genetic algorithms, and expert systems in these phases. Reference [26] investigates the interplay between AI and SE by tracing their historical evolution and identifying areas of convergence, discussing AI components like expert systems, machine learning, and natural language processing alongside SE principles such as requirements engineering and design. It analyzes factors influencing their interaction, highlighting supporting elements like automatic programming and challenges like simulating human behavior. An exploratory study by [27] advocates for standardized frameworks to ensure safe AI system development, tracing the historical evolution of AI and SE and emphasizing systematic approaches and verification techniques in software development.

Our research presents an in-depth examination of the influence of AI on SE phases and activities from 2013 to 2023, concentrating on methodologies such as machine learning (ML), deep learning (DL), and natural language processing (NLP) and their contributions to enhancing efficiency and accuracy throughout SE activities like defect prediction, code recommendation, and maintenance. The investigation explores how AI tools improve the accuracy of SE activities (e.g., refining predictions, minimizing errors) and efficiency (e.g., automating tasks, lessening manual effort, optimizing resource management) while also tackling issues such as model interpretability and ethical dilemmas.

Collectively, these studies provide complementary perspectives on AI's function in SE: our research highlights efficiency and accuracy across stages, [20] offers a broad mapping and advocates for more thorough integration, [11] centers on trends, metrics, and the predominance of ML, [21] investigates historical progress and the collaborative possibilities of AI and SE, the analysis of SLRs in SE [22], and the taxonomy for categorizing AI applications in software systems [23], the exploration of AI in automating programming, testing, planning, and project effort estimation [24], the investigation of AI techniques in software defect

prediction [25], the mapping of AI techniques to specific SE phases [1], the examination of the evolving relationship between AI and SE [26], and the emphasis on the need for standards in both SE and AI [27], our research uniquely focuses on the evolution and application of AI techniques to enhance efficiency and/or accuracy of SE phases, offering insights into the transformative changes and future directions in the field.

Overall, our research contributes valuable insights into the integration of AI in SE, emphasizing the need for interpretable AI models, exploring novel AI paradigms, and addressing ethical and societal implications to ensure the responsible deployment of AI technologies in SE. By focusing on a specific timeframe and providing a structured understanding of AI's contributions to SE, our research adds to the existing body of knowledge on the intersection of AI and SE, paving the way for future advancements in the field.

III. CONCEPTUAL FRAMEWORK AND METHODOLOGY

In the subsequent sections, we describe our research methodology and conceptual framework. The Methodology section outlines the systematic approach employed in gathering and analyzing data, providing the steps to ensure rigor and reliability in our study. Following this, the Conceptual Framework section explains the theoretical underpinnings and guiding principles shaping our investigation, providing a structured lens through which to interpret our findings. These subsections offer a comprehensive insight into the methodological and theoretical foundations underpinning our research endeavor.

A. METHODOLOGY

Systematic mapping studies offer a comprehensive overview of a particular research domain through classification. These studies involve an in-depth exploration of existing literature to assess the coverage of multiple topics, publication frequency, research trends, and relevant publication venues. In the present study, the systematic mapping process adheres primarily to the guidelines proposed by [28]. Following these guidelines specific to SE [28], the essential steps in the systematic mapping study encompassed defining research questions, searching for pertinent papers, screening the papers, keywording abstracts, extracting data, and mapping. Brereton, Kitchenham, and Budgen are well-known researchers in SLRs on SE [22], [29], [30], [31]. [29] introduces a template for creating a case study protocol in SE, aiming to enhance the rigor of case studies. The template is based on established case study methodologies. It ensures unbiased, auditable, and rigorous reviews across various disciplines, including SE, social sciences, economics, and nursing. In another similar study, a protocol for a tertiary study of SLRs and Evidence-Based Guidelines in IT and SE is proposed, emphasizing the adoption of Evidence-Based SE (EBSE) within the SE research community. The study aims to promote the application of an Evidence-Based approach in SE research and practice [30]. Another study compares the results of two SLRs conducted independently by novice researchers on unit testing methods with an expert literature review by Juristo and colleagues. The research investigates the repeatability of SLRs in SE, focusing on the stability of results [31]. Kitchenham conducted a tertiary study on systematic literature reviews in SE to identify the number of SLRs published, the research topics addressed, and the limitations of current SLRs. The study compared different sets of SLRs, discussed the quality assessments of papers, and highlighted the impact of consensus and minority reports in assessing quality [22].

Systematic Literature Reviews (SLRs) are typically conducted through a structured approach consisting of eight phases with five core phases, as shown in Figure 1. Each phase contributes distinct outcomes, ultimately culminating in creating a systematic map. The initial phase entails formulating precise research inquiries to define the scope of investigation, aiming to offer a comprehensive overview and ascertain the extent and diversity of research findings within the chosen domain. Identification of search terms is the third step after data sources are determined. The primary studies involve systematically searching scientific databases and manually reviewing conference proceedings and journal articles. Screening these studies for relevance ensures that only pertinent literature addressing the research questions is included. Abstract keywording facilitates the development of an efficient classification schema informed by existing studies. During data extraction and mapping, relevant articles are categorized according to the established schema, and studies that do not directly contribute to quality assessments or answer the research questions are excluded. Finally, studies are converted to data, become rich with synthesis and interpretation, and results are reported, thereby completing the systematic map.

For the study, several types of categorization of AI were considered, such as *capabilities* (weak AI [32], [33], strong AI [32], [33], general AI [34], superintelligent AI [35]), *functionalities* (reactive [36], memory-based [37], self-aware AI [38]), *application areas* (expert systems [39], natural language process [40], computer vision [41], robotics [42], machine learning [43], optimization [32], [44]), *learning techniques* (supervised [32], [43], unsupervised [32], [43], semi-supervised [32], [43], reinforcement [44], [45]), *methodologies* (symbolic AI [32], [36], connectionist AI [37], [44], evolutionary algorithms [32], [44]), *deployment and usage* [32] (cloud AI, edge AI), *human interaction* (human interacted [32], [44], fully automated [42], [45]).

This study selected *the application area* categorization of AI for further analysis. Within the scope of this study, a wide range of AI techniques or methods in five high-level categories, namely, (i) Machine Learning (ML), (ii) Deep Learning (DL), (iii) Optimization Algorithms (OA), (iv) Natural Language Processing (NLP), and (v) Expert Systems (ES) were considered. These techniques are illustrated in Figure 2.

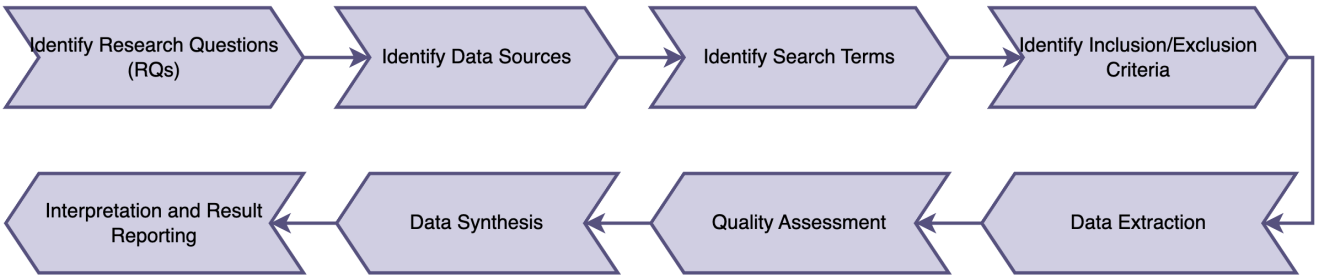


FIGURE 1. The SLR steps used in the study.

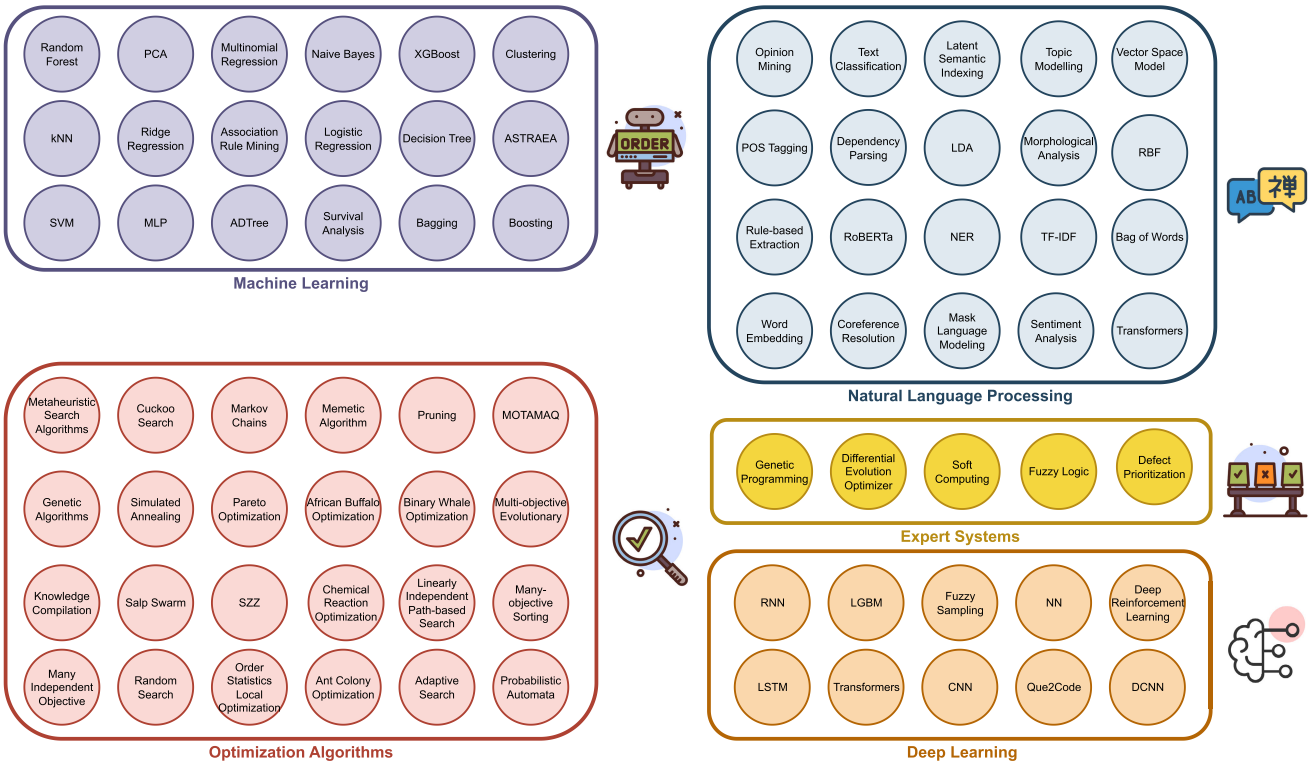


FIGURE 2. The illustration of the AI techniques or methods considered in this study.

In addition, the study refers to the seven phases of SE: I) planning, II) requirement engineering, III) design, IV) development, V) testing, VI) deployment, and VI) maintenance. These SE phases are also illustrated in Figure 3.

The following section explains the major elements of the methodology. First, research questions are expressed, data sources are shown, and search terms are presented with inclusion and exclusion criteria. Secondly, a conceptual framework shows SE phases, AI techniques, and methods.

1) RESEARCH QUESTIONS

This section begins by clarifying the distinct research questions that steer our SLR, aiming to define the breadth and emphasis of our study in the realm of SE. The primary research question for this systematic exploration is: "Does AI impact the SE phases and the related activities?". This main research question (RQ) is divided into four secondary questions. RQ1 focuses on the effects of AI across the entire field of SE, while RQ2 examines the consequences of AI on the different SE phases. RQ3 and RQ4 then highlight the effects of specific AI tools and techniques on the accuracy and/or efficiency of SE activities.

RQ1: Which algorithm classifications are applied more frequently than others for the SE domain? This question aims to identify which algorithms or methods are most used in SE. Understanding the popularity and prevalence of specific methods can highlight trends and preferences in the field. Additionally, this knowledge can guide researchers and practitioners in focusing on the most impactful and widely adopted AI techniques.

RQ2: Are there any specific AI techniques or methods applied more frequently to a particular SE phase than others? This investigates the relationship between AI methods and various SE Phases. By examining this relationship, we can gain insights into the suitability and effectiveness of certain AI techniques used for specific SE phases.

RQ3: To what extent specific AI techniques or methods positively and significantly affect the ''efficiency'' of SE phases and associated activities? This research question investigates the impact of AI techniques on enhancing the efficiency of SE phases and associated activities. Efficiency might include reduced development time, streamlined workflows, and optimized resource utilization. For instance, machine learning can predict defects early, reducing testing and debugging time, while natural language processing automates requirement classification, speeding up planning and design. Optimization algorithms can enhance resource allocation, ensuring effective use of development efforts. By evaluating these contributions across SE phases, we aim to understand how AI drives efficiency improvements in SE.

RQ4: To what extent specific AI techniques or methods positively and significantly affect the ''accuracy'' of SE phases and associated activities? This research question examines the impact of AI techniques on the precision and reliability of SE phases and associated activities. It investigates how AI methods can reduce errors, enhance code quality, and improve testing accuracy. For example, machine learning algorithms can predict and prevent bugs, while deep learning models can assist in more accurate code reviews. Natural language processing can improve requirement specifications and reduce ambiguities. By evaluating these effects, we aim to understand how AI contributes to higher accuracy and reliability in various SE phases.

2) DATA SOURCES

This section provides details of the various data sources utilized to gather literature for our systematic mapping study, focusing on identifying studies that explore the impact of AI techniques or methods on phases of SE. This search was conducted using a tool called ''Dimensions.ai,'' a linked research web AI database containing access to resources from major databases such as IEEE Xplore, ACM Digital Library, Scopus, and Web of Science. Among the various databases selected for sourcing literature, Dimensions stands out due to its comprehensive coverage. As of December 2023, Dimensions contained over 140 million publications, approximately 30% more than other comparable databases. This extensive repository dramatically enhances our ability to conduct a thorough analysis by providing access to a broader array of research outputs relevant to the impact of AI techniques or methods on SE phases.

3) SEARCH TERMS

A carefully curated set of search terms was developed, targeting publications that examine the intersection of AI techniques or methods and SE phases. Identifying key search terms is imperative for thoroughly searching pertinent studies. Kitchenham et al. [46] proposed using Population, Intervention, Comparison, and Outcome (PICO) viewpoints. These viewpoints have gained widespread acceptance and application in several SLRs. The PICO terms pertinent to the current context encompass the following: Population (P): primary studies in SE; Intervention (I): AI techniques or methods; Comparison (C): Techniques, Methods, Contribution, Dataset, Performance metric, and SE phases; and Outcome (O): Classification of the types and combination of AI techniques or methods applied in SE phases.

A comprehensive search string was formulated based on the PICO structure to ensure consistency across multiple databases. This approach aims to facilitate a systematic and extensive exploration of relevant literature in the field. The utilization of PICO criteria not only aids in refining the search scope but also provides a structured framework for identifying and categorizing pertinent studies in SE and AI. Based on the PICO structure, the generic search string given in Table 1 was created to maintain the consistency of the search across multiple databases.

4) INCLUSION AND EXCLUSION CRITERIA

The inclusion criteria for this study encompasses peer-reviewed articles sourced from journals, with a specific focus on articles describing the integration of AI techniques or methods within various SE phases (based on the classification of [11]). Furthermore, the temporal scope is confined to articles published between 2013 and 2023. These criteria aim to ensure the reliability and credibility of the selected literature while emphasizing recent developments in the intersection of SE and AI. Conversely, exclusion criteria encompass articles that fail to meet the stipulated inclusion criteria and those available in abstract or presentation formats.

Additionally, articles in languages other than English are excluded to maintain linguistic consistency and facilitate comprehensive analysis. Furthermore, articles presenting ambiguous or unclear results or findings are excluded from upholding the rigor of the study's synthesis and interpretation of relevant literature. This systematic approach to inclusion and exclusion criteria is designed to refine the selection process and enhance the quality and relevance of the gathered research corpus.

Based on the criteria, 4,185 papers were identified through Dimensions, of which ten highly cited papers for each year were selected for further review. A total of 110 papers were selected.

5) DATA EXTRACTION

Once the 110 papers were identified, the authors divided the manual work. Assigned papers were retrieved from their content, and then a central spreadsheet was populated with relevant data.

TABLE 1. Searching category, criteria, and generic string based on PICO structure.

Category	Generic String
Software Engineering	Software engineering phases
	Software engineering processes
	Software engineering practices
	Software planning
	Software requirement engineering
	Software analysis
	Software design
	Software development
	Software testing
	Software deployment
	Software training
	Software handover
	Software maintenance
Artificial Intelligence	Artificial Intelligence (AI)
	AI Techniques
	Machine Learning
	Classification and regression tree
	Quick, unbiased, efficient statistical tree (C5.0)
	Chi-squared Automatic Interaction Detection Trees
	Random Forest
	Support vector machine
	Nearest Neighbor Analysis
	K-Means
	Hierarchical Density-Based Spatial Clustering
	Naïve Bayes Classifier
	XGBoost
	Discriminant Analysis
	Fuzzy ADH
	Natural Language Processing (NLP)
	Semantic Network
	Concept Inclusion
	BERT-based classification
	Information extraction
	Sentiment analysis
	Optimization Algorithms
	Gradient boosting algorithm
	Genetic Algorithm
	Ant Colony
	Particle Swarm Optimization
	Simulated Annealing
	Tabu search
	Iterated local search
	Variable neighborhood search
	Artificial bee colony
	Firefly algorithm
	Teaching learning-based optimization
	Sine cosine optimization
	Gray wolf optimization
	Whale optimization
	Salp swarm optimization
	Harris hawk optimization
	Computer Vision
	Scale-Invariant Feature Transform (SIFT)
	Speeded Up Robust Features (SURF)
	Viola-Jones
	Eigenfaces
	YOLO
	ResNet
	Graph Cut Optimization
	Kalman Filter
	Autoencoders
	MobileNet
	Single-shot multi-box detector (SSD)
	Deep Learning
	Multilayer perceptron
	Kohonen
	Neural Networks
	Bayesian Network
	Recurrent neural networks (RNNs)
	AlexNet
	VGGNet
	ResNet
	MobileNet
	Convolutional neural network (CNN)
	Statistical Techniques
	Principal components analysis (PCA)
	Gaussian Mixture Model (GMM)
	Kernel Density Estimation (KDE)
	Exponential smoothing
	ARIMA
	Cox regression
	Hidden Markov models (HMMs)
	Fuzzy Logic
	Fuzzy AHP
	Adaptive neuro-fuzzy inference system (ANFIS)
	Expert Systems
	Apriori algorithm

B. CONCEPTUAL FRAMEWORK

This section introduces the conceptual framework underpinning our study, focusing on integrating AI within various SE phases and associated activities and their potential impacts. The execution of the SLR relies on a multi-stage model of the SE life cycle, as widely adopted within academic SE. This model encompasses critical phases, including planning, requirement engineering, design, development, testing, deployment, and maintenance, as illustrated in Figure 3.

In this paper's context, "efficiency" refers to the ability of AI to streamline and optimize various aspects of SE phases. This includes improving the speed, resource utilization, and effectiveness of software design, requirement engineering, planning, and maintenance tasks. Efficiency in this context also includes reducing manual effort, time, and cost and enhancing SE activities' quality and reliability through AI.

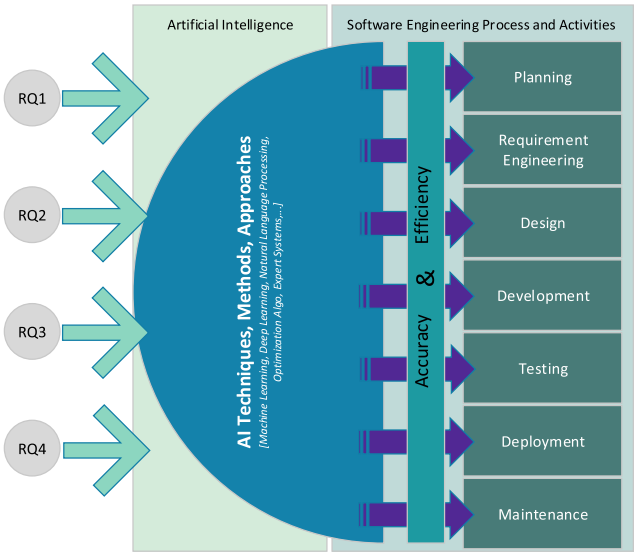


FIGURE 3. The illustration of the proposed conceptual framework.

On the other hand, "accuracy" refers to the precision and correctness of AI in predicting, analyzing, and optimizing various aspects of SE. This includes the ability of AI models to provide reliable and consistent results in tasks such as defect prediction, software size estimation, effort estimation, and requirement classification. Accuracy in this context also encompasses the capability of AI to generate dependable and trustworthy outcomes, leading to improved decision-making and overall effectiveness in SE phases.

IV. RESULT AND DISCUSSION

This section presents the results acquired through experimentation and thoroughly examines the findings. The RQs to achieve this objective will be addressed in the subsequent subsections.

A. RQ1: WHICH AI TECHNIQUES OR METHODS ARE APPLIED MORE FREQUENTLY THAN OTHERS FOR SE?

Figure 4 illustrates the top 10 frequently used AI techniques or methods. Random Forest was identified as SE's most commonly utilized AI technique, appearing in 22 related studies. Following closely, Naïve Bayes and Support Vector Machine (SVM) claim the second spot as the second most frequently used AI techniques or methods, each featured in 15 related studies. The remaining often utilized AI techniques or methods were ranked as follows: Genetic Algorithms (8), Logistic Regression (8), Decision Trees (7), Convolutional Neural Networks (6), Long-Short Term Memory (6), Neural Networks (6), and Multilayer Perceptrons (5). According to this experimental result, it is safe to assert that traditional ML algorithms are more frequently used in SE than DL.

Finding 1: Practitioners prefer to employ fast and simple AI solutions rather than complex ones.

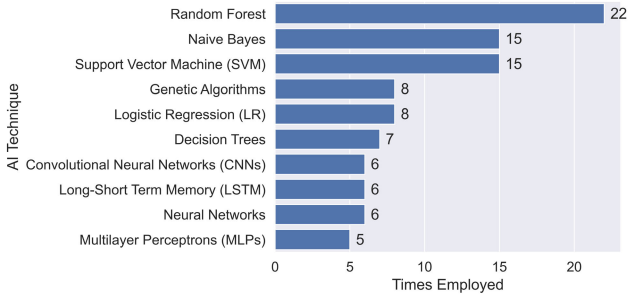


FIGURE 4. Ten most frequently utilized AI techniques or methods applied to SE.

As depicted in Finding 1, SE practitioners generally employ straightforward and fast techniques such as Naïve Bayes. It has been utilized for mapping source codes [47], commit mining [48], and refactoring prediction [49]. Further, when a sophisticated feature selection technique eliminates the number of features, Random Forest is a viable alternative, especially for effort estimation and software issue report classification.

B. RQ2: ARE THERE ANY SPECIFIC AI TECHNIQUES OR METHODS APPLIED MORE FREQUENTLY TO A PARTICULAR SE PHASE THAN OTHERS?

Figure 5 shows various AI techniques or methods in different SE phases. For the planning phase, ML emerges as a dominant theme in 29 identified studies, emphasizing its significance in tasks such as ensemble effort estimation (EEE) models and enhancing prediction accuracy in software development efforts. OA follows with ten studies highlighting their role in refining AI techniques or methods for SE planning. While DL and NLP are less prevalent, they still demonstrate relevance in 2 and 1 studies, respectively. Notably, ES was not featured in any studies, indicating a shift from traditional AI methodologies in this context. Key studies underscore ML's application in refining software size prediction and understanding the relationship between community and code smells, reflecting socio-technical issues [50], [51], [52].

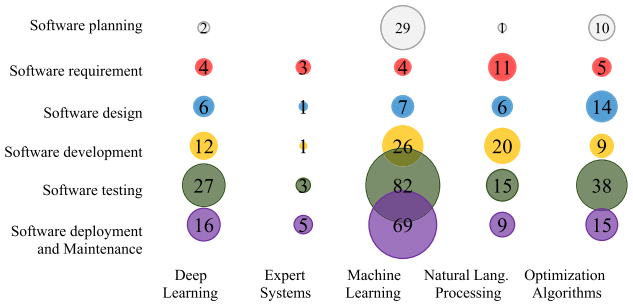


FIGURE 5. High-level AI categories and associated SE phases.

Examining the impact of AI on the Requirement Engineering phase, NLP emerges as a prominent focus, appearing 11 times in the identified literature. This prevalence underscores the significance of NLP in enhancing various aspects of software development. Notably, research demonstrates the application of NLP methods in refining vulnerability prediction, reducing manual effort, and bolstering the reliability of identifying security vulnerabilities, thereby enhancing the overall effectiveness of SE phases [53]. Additionally, NLP tools and resources detect language issues in requirements documents, addressing ambiguity, inconsistency, and incompleteness. Applying NLP techniques elevates the accuracy of requirements classification, feature identification, and sentiment analysis, offering more reliable and consistent results in SE phases [54]. Furthermore, the integration of NLP in improving the quality of user stories, defect detection, and traceability between models/artifacts is discussed, emphasizing the challenges and opportunities in achieving high precision in NLP research on user stories [55].

OA emerged as the most prevalent AI technique or method used for the Software Design phase, featured in 14 identified studies. Notably, genetic algorithms and simulated annealing within OA have significantly enhanced defect prediction accuracy, exemplifying the efficiency of evolutionary computation and probabilistic optimization [56]. Additionally, African Buffalo Optimization (ABO) highlights the utilization of nature-inspired approaches, specifically in test suite prioritization, illustrating how AI methods like OA contribute to improved SE efficiency [57]. The significance of ML is underscored by its appearance in seven studies, where techniques such as Random Forest and Binary Whale Optimization algorithm demonstrate superior performance in SE phases, as highlighted in a study focusing on classifier performance [58]. In contrast, ES features only once, suggesting a lesser prevalence in the examined literature. This comprehensive overview underscores the dominant role of OA and the broader influence of AI in shaping the SE design landscape.

The impact of AI on Software Development is profound, with ML emerging as a dominant force. ML, referenced in 26 studies, was utilized for tasks such as early detection

of security issues and vulnerabilities, enhancing software security by automating processes [59], predicting Common Vulnerability Scoring System (CVSS) metrics, and prioritizing exploits [60], and categorizing commits to improve precision in SE practices [61]. NLP followed closely behind, cited in 20 studies, highlighting its importance in various aspects of SE. DL, mentioned in 12 studies, and OA, in 9 studies, further contribute to advancing software development. ES, however, was mentioned only once, indicating their limited application in comparison [59], [60], [61].

In the Software Testing phase, the impact of AI is notably profound, as shown in Figure 5. The ML emerged as a predominant theme in the identified studies, appearing 82 times, highlighting its prevalence and significance. OA followed closely with 38 mentions, followed by DL in 27 studies and NLP in 15. ES still contributed to the discourse while being less prevalent, with only three appearances. Reference papers underscore the transformative influence of ML on SE phases, emphasizing improved dependability and consistency in outcomes. Leveraging model-agnostic techniques, such as LINE-DP, enhances accuracy, leading to targeted allocation of Software Quality Assurance resources and ultimately elevating the quality and reliability of SE activities. The utilization of AST n-grams as a bottom-up approach has shown promise in identifying code features from the perspective of SE's deployment and maintenance phases. ML emerged as the most frequently referenced AI technique, with 62 occurrences highlighting its pervasive influence. DL is closely followed with 16 mentions, while OA and NLP appear in 27 and 9 studies, respectively. ESs were mentioned in 5 studies. Noteworthy applications of ML in SE include optimizing Latent Dirichlet Allocation (LDA) hyperparameters for improved decision-making and effectiveness and enhancing precision in tasks such as feature location and traceability recovery [62]. Additionally, a data-driven approach utilizing ML models accurately predicted GitHub project maintenance activity, evaluating precision and correctness through standard metrics [63]. Supervised ML algorithms have significantly advanced fake review detection in app stores, leading to enhanced decision-making and operational effectiveness [64].

Regarding the relation between RQ1 and RQ2, we can conclude that the most used AI techniques, such as Random Forest and Naïve Bayes, are generally employed in the requirement and design phases rather than the testing phase. Heuristics such as the genetic algorithm are much more effective in that phase. For instance, genetic algorithms and mutation operators are common among test case generation techniques. On the other hand, the projection of the software budget is done via sophisticated decision support systems rather than expert opinions.

Finding 2: Machine learning plays a key role in planning by improving ensemble effort estimates and boosting prediction accuracy. Natural language processing (NLP) is crucial in requirement engineering by enhancing vulnerability prediction and requirements analysis. In software testing, machine learning significantly improves quality assurance. In deployment and maintenance, it continues to be widely used to optimize decision-making and predict maintenance needs [65], [66], [67].

C. RQ3: TO WHAT EXTENT DO SPECIFIC AI TECHNIQUES OR METHODS POSITIVELY AND SIGNIFICANTLY AFFECT THE "EFFICIENCY" OF SE PHASES AND ASSOCIATED ACTIVITIES?

Among the 110 papers examined, a subset of 36 papers was identified to positively influence the efficiency of SE phases and related activities. These investigations have probed the confluence of AI methodologies and SE principles, emphasizing augmenting efficiency throughout distinct SE phases. Table 2 maps the identified papers, AI employed, and their consequential impact on efficiency across various SE phases and activities. Specifically, 16 papers implemented OA, followed by six papers employing ML and five papers centering on NLP.

Paper [57] applies ABO to streamline test case prioritization and selection, demonstrating how nature-inspired AI methods, such as those inspired by African buffalos, positively impact the efficiency of software testing processes. Similarly, the [68] introduces BDDSampler, an AI method leveraging binary decision diagrams for optimizing the sampling process of highly configurable systems, illustrating how AI can reduce manual effort, time, and cost while improving reliability in SE activities. The [69] discusses the effectiveness of evolutionary algorithms in generating unit test suites for code coverage, emphasizing the improved accuracy and efficiency in software testing. Similarly, [70] explores seeding strategies to optimize automated unit test generation, aligning with enhancing efficiency in software testing processes.

Reference [89] investigates whether tuning data miners for software defect prediction significantly impacts their performance. The study uses differential evolution for tuning and evaluated its effect on the performance of various data miners like CART, Random Forest, and Logistic Regression in defect prediction tasks.

Reference [92] introduces a strategy for generating combinatorial test suites using the Cuckoo Search algorithm to optimize test case generation for software functional testing. This approach enhances efficiency by reducing the number of test cases needed while maintaining effective fault detection. Reference [94] tests a memetic algorithm to integrate local search with global search techniques to optimize the generation of complete test suites for software testing. This approach aims to improve the speed and effectiveness of test case generation, addressing efficiency in SE phases by reducing the manual effort and time required for test suite development.

Reference [96] discusses the use of genetic algorithms for optimizing test suites in software testing, aiming to improve

TABLE 2. Mapping of High-level AI applied, and efficiency gained in various SE phases and associated activities.

Reference	AI Technology	Efficiency in SE Activities	SE Phases
[71]	**ES**	Streamlined test prioritization	Software Testing
[72]	**ES, OA**	Improved bug fixing	Software Testing
[73]		Testing efficiency improvement	Software Testing
[74]	**ML**	AI-based code review	Software Development, Software Testing, Software Maintenance
[75]		Efficient configuration	Software Development, Software Testing, Software Maintenance
[76]		Defect prediction improvement	Software Testing
[77]		Cost estimation efficiency	Software Testing
[78]		Triage efficiency improvement	Software Testing
[79]		Bug report priority efficiency	Software Testing
[80]	**ML, DL**	Automated code assessment	Software Maintenance
[81]		Efficient GUI conversion	Software Design, Software Development
[82]		Privacy-preserving defect prediction	Software Testing
[60]	**ML, DL, LP**	Streamlined assessment	Software Testing, Software Maintenance
[59]	**ML, DL, OA**	Enhanced security	Software Requirement, Software Maintenance
[83]	**NLP**	Automated UML conversion	Software Design
[84]		Automated defect correction	Software Requirement
[85]		Efficient code search	Software Development
[86]		Task extraction efficiency	Software Requirement Software Development
[87]		Refactoring identification	Software Maintenance
[57]	**OA**	Streamlined testing	Software Testing
[68]		Efficient sampling	Software Testing
[69]		Test suite optimization	Software Testing
[70]		Enhanced test generation	Software Testing
[88]		Proactive project rescheduling	Software Planning
[89]		Improved defect prediction	Software Testing
[90]		Feature selection optimization	Software Development
[91]		Interaction testing efficiency	Software Testing
[92]		Test suite generation efficiency	Software Testing
[93]		Re-modularization efficiency	Software Development, Software Maintenance
[94]		Test suite generation efficiency	Software Testing
[95]		Efficient crowdsourcing tasks	Software Development
[96]		Testing efficiency improvement	Software Testing
[97]		Large-scale project efficiency	Software Design
[98]		Test case generation efficiency	Software Testing
[99]		Development efficiency improvement	Software Planning, Software Design
[71]	**ES**	Streamlined test prioritization	Software Testing

coverage while keeping the size small. This approach enhanced efficiency by streamlining the test generation process, reducing manual effort, and improving resource utilization. Reference [96] investigates EvoSuite, a tool employing Search-based Software Testing (SBST) to generate test suites. The tool significantly improves the efficiency of testing processes by automating test case generation and optimizing for both code coverage and finding violations of automated oracles.

Reference [100] focuses on optimizing several SE objectives, such as improving the structure of software packages, minimizing code changes, and maximizing consistency with development history. By employing NSGA-III (Non-dominated Sorting Genetic Algorithm III), the paper demonstrates how AI methods could significantly enhance the process of software modularization, making it more efficient in resource utilization and effectiveness. Reference [99] explores the application of Ant Colony Optimization (ACO) for selecting software requirements, a process crucial for efficient software development. The study demonstrates how, as a metaheuristic algorithm, ACO can efficiently handle the requirements selection problem by optimizing multiple conflicting objectives like cost and customer satisfaction.

Reference [90] introduces IBED, an algorithm combining Indicator-Based Evolutionary Algorithm (IBEA) and Differential Evolution (DE), to address optimal feature selection in software product lines. This approach aims to improve solutions' correctness (accuracy), enhancing diversity and reducing computational time (efficiency). Reference [95] introduces a novel multi-armed bandit (MAB) model, the bounded MAB, and an associated algorithm bounded ϵ-first for optimizing expert crowdsourcing tasks under budget and quality constraints. This model efficiently tackles the selection and assignment of tasks to workers within a budget, aiming to maximize overall utility.

In a different context, paper [88] highlights how AI-based automated code review systems improve software design efficiency by addressing code smells, demonstrating that these methods streamline various SE activities. Reference [97] studies using variability models and languages, particularly Kconfig and CDL, in large-scale software projects for optimizing and streamlining SE phases, such as design, planning, and maintenance.

Reference [74] highlights how ML-based automated code review systems have improved the efficiency of software design by identifying and addressing code smells, reducing manual effort, and enhancing the overall effectiveness of the design process. Reference [75] uses FLASH, a sequential model-based method, to expedite finding optimal configurations for software systems. This method significantly streamlined the exploration of configuration space, thereby improving speed and resource utilization. Reference [76] proposes a novel approach called Heterogeneous Defect Prediction (HDP) that predicts defects across projects with heterogeneous metric sets. The methodology involved metric selection and matching, demonstrating that HDP predictions are often comparable or superior to within-project defect predictions (WPDP).

Reference [77] discusses how various preprocessing techniques, such as feature selection, scaling, and missing-data treatments, can influence the performance of ML models. Reference [78] focuses on improving the efficiency of bug triage in software development. It addresses the problem of large-scale and low-quality bug data in bug repositories, proposing a novel approach to reduce the data scale and improve the quality of bug data. Similarly, [79] introduces an ML-based method that considers multiple factors like text content, author, related reports, severity, and product to automate the priority assignment of bug reports.

Reference [83] uses an NLP approach to convert user stories into Unified Modeling Language (UML) automatically using case diagrams. This approach significantly enhances the efficiency of the software development process by automating the transformation of user requirements into formal models. Reference [84] introduces a Quality User Story (QUS) framework and the Automatic Quality User Story Artisan (AQUSA) tool, leveraging NLP techniques. These tools are designed to identify and suggest corrections for quality defects in user stories, an essential element in agile development processes. Reference [101] introduces the *QECK* (Query Expansion based on Crowd Knowledge) method, which improves code search algorithms by incorporating expansion words derived from crowd-sourced knowledge on *Stack Overflow*.

The testing and maintenance phases implement AI, with 27 and 12 studies, respectively (Table 3). ML and OA are the AI types most used in software development, with 18 and 22 studies, respectively. Software planning is mainly related to scheduling that needs to be optimized, and two studies include all related studies. The ES is used only in the testing phase.

In the study [86] develops an approach that improves accuracy by precisely identifying relevant tasks in the documentation, thus aligning to provide reliable and consistent AI-based predictions and analyses in SE. Additionally, the paper enhances efficiency by streamlining the process of navigating complex software documentation, reducing manual effort and time required for developers to find relevant information.

TABLE 3. Mapping of High-level AI and SE phases in the context of efficiency.

Development Phase	DL	ES	ML	NLP	OA	Total
Software Design	1	-	1	1	2	5
Software Development	1	-	3	2	3	9
Software Maintenance	3	-	5	2	2	12
Software Planning	-	-	-	-	2	2
Software Requirement	1	-	1	2	1	5
Software Testing	2	4	8	1	12	27
Total	8	4	18	8	22	60

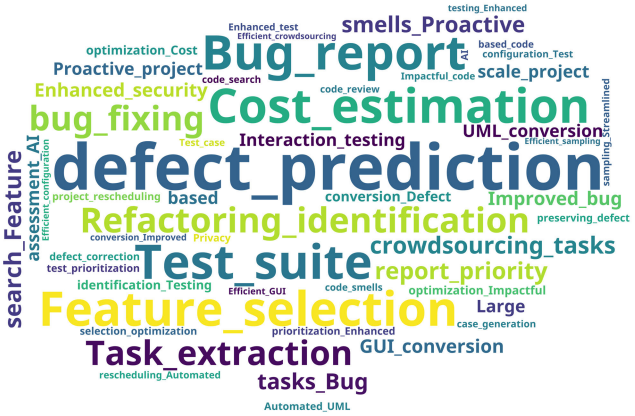


FIGURE 6. The generated word cloud on the effect of the efficiency in the SE.

On the other hand, [102] proposes an AI-driven technique called *Methodbook*, which employs Relational Topic Models (RTM) to identify refactoring opportunities for removing the "Feature Envy" bad smell from source code. Based on Table 1, a word cloud was generated in Figure 6 to highlight efficiency in various phases of SE.

The 36 papers were identified, integrating AI techniques and methods like OA, ML, and NLP to enhance the efficiency of SE phases. Noteworthy studies include ABO for test case prioritization, BDDSampler for system sampling, and evolutionary algorithms for unit test generation. Other approaches optimize defect prediction, test suite generation, and software re-modularization, enhancing SE efficiency and effectiveness.

Finding 3: Our finding shows that AI techniques like OA, ML, and NLP enhance SE efficiency. Key studies include ABO for test case prioritization, BDDSampler for system sampling, and evolutionary algorithms for unit test generation. These approaches optimize defect prediction and software re-modularization.

D. RQ4: TO WHAT EXTENT DO SPECIFIC AI TECHNIQUES OR METHODS POSITIVELY AND SIGNIFICANTLY AFFECT THE "ACCURACY" OF SE PHASES AND ASSOCIATED ACTIVITIES?

Among the corpus of 110 papers scrutinized, 74 were identified as having a positive impact of AI in augmenting the accuracy of SE phases and associated activities.

These identified studies discuss the convergence of AI and SE, specifically emphasizing elevating accuracy throughout diverse SE phases and activities. Table 4 systematically presents a detailed explanation of these studies and a thorough mapping of AI techniques and methods to their effects on accuracy in different SE phases and activities. Notably, 20 identified papers exhibit the employment of ML techniques and methodologies, followed by 13 papers employing DL, 8 papers centering on NLP, and 7 papers devoted to implementing OA.

ML techniques have significantly impacted various phases of SE, positively affecting accuracy and decision-making processes. Numerous studies across different years have demonstrated the efficiency of ML algorithms in various SE tasks, ranging from defect prediction to effort estimation and code smell detection.

One significant area where ML has had a major impact is predicting software system defects. Research, such as the paper by [67] that introduces the DP-ARNN method and the study by [103] that explores class rebalancing techniques, has improved the accuracy and reliability of defect prediction. These improvements were made using ML algorithms to automatically create syntactic and semantic features and apply techniques like oversampling, undersampling, SMOTE, and ROSE to enhance performance metrics.

Moreover, ML techniques have been successfully used to predict software refactoring, as noted in the [104]. Supervised ML algorithms have shown reliability and consistency in identifying refactoring opportunities, aiding in more efficient software maintenance and development. Besides defect prediction and refactoring, ML is essential for detecting code smells. Reference [105] shows how data balancing techniques improve the accuracy of code smell detection, enhancing the reliability and consistency of the results. Likewise, the [106] highlights the precision and accuracy of AI-driven tools in detecting code smells and analyzing software faults, enabling engineers to prioritize refactoring efforts effectively.

Effort estimation, a vital part of SE, has also improved thanks to ML techniques. The study in [50] utilizes ML techniques to boost the accuracy of the estimation, especially by using ensemble models. By applying ML, these studies increased the precision and reliability of predictions, greatly enhancing our understanding and improving the accuracy of effort estimation in SE, as shown in the study referenced in [107].

Furthermore, ML techniques have been employed in various other SE tasks, including test case selection and prioritization, measurement of maintenance activity in GitHub projects, fake review detection in app stores, and predicting web application vulnerabilities.

Integrating DL techniques in various aspects of SE has undoubtedly led to significant improvements in the accuracy of crucial phases. The papers provided offer a comprehensive view of how DL methods contribute to defect prediction, code recommendation, program analysis, code summarization, clone detection, error detection, plagiarism detection, cybersecurity, and project estimation.

Defect prediction is an essential part of SE. Reference [108] introduces SG-Trans, a system that uses machine learning techniques to predict defects and estimate software size accurately. This method improved decision-making by offering more precise information about potential problems during development. Additionally, the [109] highlights the effectiveness of deep learning in managing class imbalance issues, which further increased the accuracy of defect prediction.

Code recommendation is another important area discussed in the [110] paper introducing Que2Code. Deep learning (DL) provides accurate code snippets using models that predict the quality of questions. This helps with code completion and improved decision-making by suggesting relevant and high-quality code snippets. The accuracy of neural program models in predicting method names in source code, as examined in the [111] paper, demonstrates the versatility of neural network models. This has greatly influenced program analysis tasks, enhancing decision-making by offering more precise insights into the code's functionality. The [112] paper discusses the use of reinforcement learning to enhance the accuracy of source code summarization. This improvement helps developers make better decisions by offering clear and precise summaries, which facilitate the understanding and maintenance of complex codebases. The paper also explores clone detection, highlighting the use of attention mechanisms and hybrid code representations. This method greatly enhances the precision and accuracy of clone detection, thereby improving decision-making by more effectively identifying and managing code duplications.

The [115] addresses error detection in source code using an attention-based language model with deep LSTM neural networks, which improves accuracy. This method increases the reliability and trustworthiness of SE tasks by ensuring more dependable error detection. Reference [116] emphasizes the accuracy and correctness of AI models in detecting software plagiarism, identifying malware threats, and classifying software similarity. This has directly influenced cybersecurity threat detection, enhancing the reliability of software systems.

The [117] discusses the detection of admitted technical debt (SATD), demonstrating that AI techniques, such as Convolutional Neural Networks (CNN), improve accuracy. This method aids in better decision-making and enhances software quality by identifying and addressing technical debt early in development. Similarly, [118] highlights bug detection in SE, showing that deep learning and attention-based neural networks increase accuracy, leading to more reliable and consistent results. This approach advances the decision-making process by offering more precise insights into the presence of bugs in the code.

In software defect prediction, the study referenced as [119] utilizes attention-based recurrent neural networks (ARNN)

TABLE 4. Mapping of High-level AI applied, and accuracy gained in various SE phases and associated activities.

Reference	AI Technology	Accuracy Applied on SE Activities	Software Engineering Phase
[108]	DL	Accuracy in Code Summarization	Software Development
[109]		Accuracy in Software Defect Prediction	Software Development, Software Testing
[110]		Accuracy in Code Search and Recommendation	Software Development
[111]		Method Prediction Accuracy	Software Development, Software Maintenance
[112]		Code Summarization Precision	Software Development, Software Maintenance
[113]		Defect Prediction Precision	Software Testing
[114]		Code Clone Detection Precision	Software Development, Software Maintenance
[115]		Error Detection Accuracy	Software Testing
[116]		Plagiarism Detection Precision	Software Testing, Software Maintenance
[117]		SATD Detection Accuracy	Software Development
[118]		Bug Detection Accuracy	Software Testing, Software Maintenance
[119]		Defect Prediction Precision	Software Testing, Software Maintenance
[120]		Effort Estimation Enhancement	Software Planning, Software Design
[54]	DL, ML	Vulnerability Detection Accuracy	All phases
[121]	DL, ML	Maintainability Prediction Accuracy	Software Maintenance
[122]	DL, NLP	Defect Prediction Accuracy	Software Testing, Software Maintenance
[123]	ES	Improved Goal Modeling Accuracy	Software Requirement
[100]	ES	Probabilistic Automata Modeling	All phases
[124]	ES	Parameter Synthesis Accuracy	Software Requirement
[125]	ES, ML, DL	Enhanced Defect Prediction Precision	Software Testing, Software Maintenance
[126]	ML	Accuracy in Software Defect Prediction in Mobile Apps	Software Testing
[61]		Refactoring Classification Precision	Software Maintenance
[65]		Test Case Selection Accuracy	Software Testing
[127]		Defect Prediction Accuracy	Software Testing
[50]		Effort Estimation Accuracy	Software Planning
[104]		Refactoring Prediction Accuracy	Software Development, Software Maintenance
[105]		Code Smell Detection Accuracy	Software Development, Software Testing
[63]		Maintenance Activity Prediction Accuracy	Software Maintenance
[64]		Fake Review Detection Accuracy	Software Maintenance
[128]		Defect Prediction Precision	Software Testing, Software Maintenance
[103]		Defect Prediction Improvement	Software Testing, Software Maintenance
[129]		Improved Defect Prediction Precision	Software Testing
[52]		Software Size Estimation Enhancement	Software Planning, Software Design, Software Testing
[130]		Improved Vulnerability Prediction	Software Testing
[131]		Reduced Metric Set, Maintained Accuracy	Software Testing
[132]		Code Smell Detection Accuracy	Software Development, Software Testing, Software Maintenance
[133]		Security Vulnerability Prediction	Software Requirement, Software Development, Software Maintenance
[134]		Process Metrics for Defect Prediction	Software Testing
[106]		Code Smell Detection Impact	Software Testing
[107]		Software Effort Estimation Enhancement	Software Planning
[135]	ML, DL	Defect Prediction Meta-Analysis	Software Testing
[66]	ML, ES	Defective Line Prediction Precision	Software Testing
[136]	ML, ES, DL	Improved Defect Prediction Accuracy	Software Testing
[137]	ML, NLP	Accuracy in Fairness Testing in NLP Systems	Software Testing
[138]	ML, NLP	Requirement Traceability Accuracy	Software Design, Software Development
[58]	ML, OA	Accuracy in Software Fault Prediction	Software Testing
[139]	ML, OA	Bug Prediction with Feature Selection	Software Testing, Software Maintenance
[107]	ML, OA	Software Effort Estimation Optimization	Software Planning
[140]	NLP	Accuracy in Automatic Test Case Creation	Software Testing
[141]		Precision and Correctness	Software Development
[142]		Requirements Analysis Precision	Software Requirement
[55]		User Stories Quality Improvement	Software Requirement, Software Development
[62]		LDA Optimization Precision	Software Maintenance
[143]		Sentiment Analysis Tool Disagreement	Software Development
[144]		Improved Vulnerability Prediction	Software Planning, Software Design, Software Testing
[102]		Software Modularization Improvement	Software Maintenance
[145]	NLP, DL, ML	Accuracy in Bug Prediction in Software Development	Software Testing
[146]	NLP, ML	Accuracy in Predicting Issue Reports Objectives and Priority	Software Testing, Software Maintenance
[147]		Change Verification Precision	Software Testing
[148]		Improved Sentiment Analysis Accuracy	Software Development
[149]		Improved F1-score in Debt Identification	Software Development, Software Testing, Software Maintenance
[56]	OA	Precision and Correctness	Software Requirement, Software Design, Software Testing
[150]		Precision and Correctness in Software Config Tuning	Software Requirement, Software Maintenance
[56]		Accuracy in Recommending Metamodel Concepts	Software Design
[151]		Code Analysis Precision	Software Development, Software Maintenance
[152]		Test Case Generation Accuracy	Software Development, Software Testing
[153]		Trustworthy Self-Adaptive Systems	Software Design, Software Development, Software Maintenance
[154]		Automated Program Repair Accuracy	Software Testing, Software Maintenance
[155]	OA, DL	Accuracy in Software Fault Prediction	Software Testing
[156]	OA, ML	Defective File Prediction Precision	Software Testing
[157]		Code Recommendation Accuracy	Software Development
[158]	OA, NLP	Personalized Change Classification	All Phases

to bolster accuracy, thereby augmenting decision-making effectiveness in defect prediction endeavors. Similarly, the research cited as [120] concentrates on agile software project estimation, employing deep learning architectures such as

Long Short-Term Memory (LSTM) and Recurrent Highway Networks (RHN) to refine the accuracy of effort estimation.

Natural Language Processing (NLP) techniques have progressively assumed a pivotal role in diverse stages of SE, markedly improving precision and efficacy across various tasks.

The [54] highlights the crucial importance of Natural Language Processing (NLP) in analyzing language. Utilizing NLP methods like recognizing named entities and causal relationships improves the precision of requirement analysis, forming a more robust basis for software development processes. Another study [55] further stresses the importance of NLP in tasks related to requirements. It investigates user stories and identifies defects. The goal is to enhance defect detection accuracy and improve the ability to trace links within software systems by employing NLP techniques. The integration of named-entity recognition and causal relationship recognition demonstrates how AI-driven methods can enhance the accuracy of defect identification and traceability.

NLP methods are increasingly impacting the automation of test case creation, surpassing their traditional usage in requirements engineering, as evidenced in a recent paper [140]. This research underscores the efficiency enhancements achieved through NLP-driven extraction of conditionals, showcasing NLP's capability to optimize the generation of test cases and enhance overall testing precision. Broadening the focus to encompass various SE stages, another scholarly work [141] offers an extensive view of AI's involvement in defect prediction, size and effort estimation, and requirement categorization. NLP techniques are pivotal in facilitating these activities, contributing to more precise predictions and classifications, thereby augmenting software development outcomes.

In a study on enhancing SE tasks, [62] introduces metaheuristics and surrogate metrics to improve Latent Dirichlet Allocation (LDA) for topic modeling and traceability recovery. By refining LDA with optimization techniques, the research aims to achieve more reliable results in activities like feature location, highlighting the significance of Natural Language Processing (NLP) in enhancing SE precision. Another study [143] examines the accuracy of sentiment analysis tools within SE contexts, stressing the necessity of domain-specific training. This research emphasizes the importance of tailored NLP methods to address the subtleties of sentiment analysis in software-related contexts, thereby improving the accuracy of sentiment-driven tasks in SE procedures.

Additionally, in predicting vulnerabilities, [144] emphasizes the effectiveness of combining static and dynamic analysis in a hybrid approach. Through the application of artificial intelligence techniques or methods, this investigation presents dependable outcomes in pinpointing sections of code susceptible to vulnerabilities, underscoring the role of natural language processing (NLP) in fortifying software security. Lastly, another paper introduced an automated method employing Relational Topic Models (RTM) for organizing software modules [87]. This research provides precise suggestions for restructuring by examining underlying themes and interconnections using NLP-powered RTM, thus enhancing software design and maintenance practices.

Incorporating Open Access (OA) and methodologies across different SE stages has notably enhanced precision and decision-making. Defect prediction is a particularly noteworthy domain where OA has shown significant positive effects. Reference [56] underscores the effectiveness of Genetic Algorithms and Simulated Annealing in enhancing the accuracy of defect prediction. By utilizing these optimization methods, AI models can more effectively scrutinize software codebases, pinpoint potential defects, and assist in decision-making concerning defect prioritization and mitigation strategies. Accurate defect prediction enhances software quality and streamlines resource allocation and risk management procedures.

Additionally, OAs have played a crucial role in advising software configuration adjustments, as outlined in the study referenced by [56]. Machine learning and optimization techniques can adjust software configurations to enhance system performance, reliability, and scalability. This process, driven by optimization, guarantees that software systems operate at their best across various conditions, enhancing the precision and effectiveness of decision-making concerning system configurations.

In SE, the design of metamodels, which are fundamental structures, has seen advancements due to Optimization Algorithms (OA). Reference [159] demonstrates the effectiveness of suggesting metamodel concepts during modeling through Deep Learning (DL) models. This approach showcases how optimization-driven methods aid in making better decisions in metamodel design. Using OA, particularly within DL frameworks, software engineers can progressively enhance metamodels to represent the intricate relationships within complex systems accurately. This metamodel design improvement subsequently enhances subsequent SE phases' accuracy and reliability.

Furthermore, Object-Oriented Analyses (OAs) have demonstrated their significance in code examination and classification duties, as evidenced by the Flow2Vec technique described in [126]. By prioritizing accuracy and correctness in code scrutiny, approaches guided by OAs facilitate more precise recognition and categorization of code components, thereby aiding tasks like bug identification, code restructuring, and upkeep. Test case creation, a pivotal aspect of software quality assurance, has also reaped benefits from OAs, as illustrated in the [152] article. The focus on the Improved Combined Fitness (ICF) function emphasizes the role of OAs in devising test cases that comprehensively cover essential pathways within software systems. This precision-oriented strategy for test case generation ensures exhaustive validation and authentication of software functionality, thereby augmenting overall precision and dependability.

Moreover, the creation of reliable self-adjusting software systems, as suggested by ENTRUST in the research paper [153], underscores the significance of dynamic assurance mechanisms facilitated by Open Architecture (OA) techniques. Such self-adjusting systems can have superior precision and dependability, even amidst intricate and uncertain operational settings, by consistently refining system parameters and configurations to adapt to evolving environmental circumstances and demands.

The development, testing, and maintenance phases primarily implement AI, with 23, 52, and 33 studies, respectively (Table 5). DL, ML, and NLP are the AI types most used in SE phases, with 35, 57, and 24 studies, respectively. OA was not used in software development. The ES is mainly related to decisions that could be used in the testing and requirement phases.

Finally, using probabilistic models to predict software repair actions, as explored in the [154], underscores the significance of OA in automating software maintenance processes. By analyzing large datasets of bug fixes and employing optimization-driven approaches, such as probabilistic modeling, software engineers can accurately predict and prioritize repair actions, thereby improving the efficiency and effectiveness of software maintenance activities. Based on Table 2, the word cloud presented in Figure 7 was generated to highlight accuracy in various phases of SE.

TABLE 5. Mapping of High-level AI and SE phases in the context of accuracy.

SE Phases	DL	ES	ML	NLP	OA	Total
Software Design	2	1	3	2	1	9
Software Development	8	1	8	6	-	23
Software Maintenance	10	2	14	5	2	33
Software Planning	2	1	5	1	1	10
Software Requirement	1	3	2	2	2	10
Software Testing	12	4	25	8	3	52
Total	**35**	**12**	**57**	**24**	**9**	**137**

The 74 papers showcased ML's positive impact on the phases of SE. ML advances enhance accuracy, notably in defect prediction, refactoring, and effort estimation. DL and NLP techniques boost accuracy in various SE tasks like code recommendation and vulnerability prediction. Optimization approaches contribute significantly, especially in defect prediction and software configuration tuning.

Finding 4: Our findings show that machine learning improves accuracy in defect prediction, refactoring, and effort estimation. Deep learning and NLP enhance code recommendation and vulnerability prediction, while optimization techniques aid in defect prediction and software configuration tuning.

V. CONCLUSION, LIMITATIONS, AND FUTURE WORK

This study comprehensively examines the application and impact of AI techniques across various SE phases and activities. By systematically mapping the existing literature, we aimed to classify AI techniques and analyze their impact on SE phases and associated activities. Hereby, we intend to start a discussion about to what extent do AI techniques affect SE phases.

Our research identifies and categorizes AI techniques used in different SE phases, assesses their effectiveness, and highlights existing trends and research gaps. The key contribution of our study includes a systematic classification of AI techniques applied in SE, a detailed analysis of these techniques on the efficiency and accuracy of SE phases and associated activities, and the identification of prevalent challenges and limitations in the current applications of AI in SE.

The findings from our study indicate that AI techniques significantly enhance the efficiency and accuracy of various SE phases and activities. Machine Learning (ML) and Natural Language Processing (NLP) are particularly effective in the planning, requirement engineering, and maintenance phases. AI techniques improve SE phases by automating tasks, reducing manual effort, and enhancing predictive accuracy, thereby reducing errors and improving reliability.

Several benefits of AI in SE phases and activities were identified. In the planning phase, ML techniques enhance prediction accuracy and efficiency. NLP techniques improve requirements classification, feature identification, and sentiment analysis in requirement engineering. Optimization algorithms enhance test case generation and optimization processes during testing. Deep learning techniques improve bug detection accuracy and efficiency, while AI techniques enhance bug prediction, vulnerability detection, and the automation of maintenance tasks during deployment and maintenance.

Despite the notable benefits, several challenges and limitations are identified. More empirical validation and industrial case studies are needed to provide robust evidence for AI adoption in SE. Integration challenges with existing SE tools and workflows, model complexity and interpretability issues, high implementation costs and computational requirements, ethical considerations, and the need for responsible AI integration were also highlighted. Model complexities are rather related to the calculation of algorithmic burden comes potentially from the purposes of AI operations.

This study is unique in its comprehensive approach, systematically mapping AI techniques across all SE phases and providing a detailed analysis of their impact and effectiveness. Unlike other studies focusing on specific AI techniques or SE phases, our research offers a holistic view, identifying key trends and gaps in the literature.

Future research should prioritize empirical validation and industrial case studies to substantiate AI integration in SE. Additionally, efforts should focus on enhancing AI technique compatibility with current SE tools, addressing model complexity and interpretability challenges, and reducing implementation and computational costs. Exploring new AI paradigms like explainable AI can improve transparency and reliability, while ethical considerations are essential for responsible AI integration in SE.

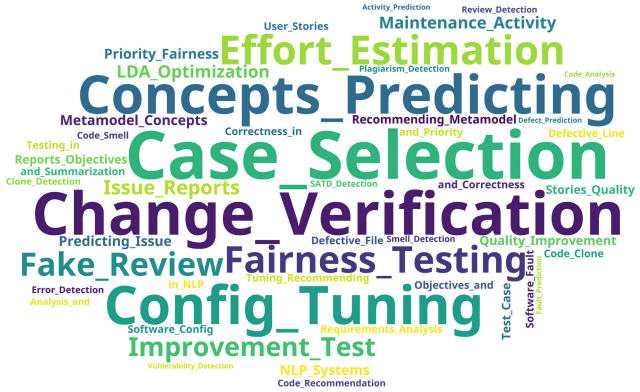


FIGURE 7. The generated word cloud on the effect of the accuracy in the SE.

In conclusion, our study provides a comprehensive overview of the impact of AI on SE phases, highlighting the significant benefits and identifying key challenges and future research directions. By addressing these challenges, the integration of AI in SE can be further enhanced, leading to more efficient and accurate SE phases.

REFERENCES

[1] H. H. Ammar, W. Abdelmoez, and M. S. Hamdi, "Software engineering using artificial intelligence techniques: Current state and open problems," in *Proc. 1st Taibah Univ. Int. Conf. Comput. Inf. Technol.*, vol. 52, 2012, pp. 1–24.

[2] S. Martínez-Fernández, J. Bogner, X. Franch, M. Oriol, J. Siebert, A. Trendowicz, A. M. Vollmer, and S. Wagner, "Software engineering for AI-based systems: A survey," *ACM Trans. Softw. Eng. Methodol.*, vol. 31, no. 2, pp. 1–59, Apr. 2022.

[3] G. Giray, "A software engineering perspective on engineering machine learning systems: State of the art and challenges," *J. Syst. Softw.*, vol. 180, Oct. 2021, Art. no. 111031.

[4] Y. Yang, X. Xia, D. Lo, and J. Grundy, "A survey on deep learning for software engineering," *ACM Comput. Surveys*, vol. 54, no. 10s, pp. 1–73, Jan. 2022.

[5] M. Savary-Leblanc, L. Burgueño, J. Cabot, X. Le Pallec, and S. Gérard, "Software assistants in software engineering: A systematic mapping study," *Software, Pract. Exper.*, vol. 53, no. 3, pp. 856–892, Mar. 2023.

[6] M. Virvou, "The emerging role of artificial intelligence in teaching 21st century skills in software engineering," in *Proc. 35th IEEE Conf. Softw. Eng. Educ. Training, CSEE&T 2023*, Apr. 2023, pp. 1–24.

[7] T. Xia, R. Shu, X. Shen, and T. Menzies, "Sequential model optimization for software effort estimation," *IEEE Trans. Softw. Eng.*, vol. 48, no. 6, pp. 1994–2009, Jun. 2022.

[8] J. Chakraborty, S. Majumder, Z. Yu, and T. Menzies, "Fairway: A way to build fair ML software," in *Proc. 28th ACM Joint Meeting Eur. Softw. Eng. Conf. Symp. Found. Softw. Eng.*, Nov. 2020, pp. 654–665.

[9] R. Sulaiman, D. A. Jawawi, and S. Halim, "Cost-effective test case generation with the hyper-heuristic for software product line testing," *Adv. Eng. Softw.*, vol. 175, Jan. 2023, Art. no. 103335.

[10] V. Garousi, A. Rainer, P. Lauvas, and A. Arcuri, "Software-testing education: A systematic literature mapping," *J. Syst. Softw.*, vol. 165, Jul. 2020, Art. no. 110570.

[11] M. Barenkamp, J. Rebstadt, and O. Thomas, "Applications of AI in classical software engineering," *AI Perspect.*, vol. 2, no. 1, p. 1, Dec. 2020.

[12] A. D. Carleton, E. Harper, T. Menzies, T. Xie, S. Eldh, and M. R. Lyu, "The AI effect: Working at the intersection of AI and SE," *IEEE Softw.*, vol. 37, no. 4, pp. 26–35, Jul. 2020.

[13] L. Yu, "Project engineering management evaluation based on GABP neural network and artificial intelligence," *Soft Comput.*, vol. 27, no. 10, pp. 6877–6889, May 2023.

[14] S. Hameed, Y. Elsheikh, and M. Azzeh, "An optimized case-based software project effort estimation using genetic algorithm," *Inf. Softw. Technol.*, vol. 153, Jan. 2023, Art. no. 107088.

[15] N. Sreekanth, J. Rama Devi, K. A. Shukla, D. K. Mohanty, A. Srinivas, G. N. Rao, A. Alam, and A. Gupta, "Evaluation of estimation in software development using deep learning-modified neural network," *Appl. Nanoscience*, vol. 13, no. 3, pp. 2405–2417, Mar. 2023.

[16] H. Nakahara, A. Monden, and Z. Yucel, "A simulation model of software quality assurance in the software lifecycle," in *Proc. IEEE/ACIS 22nd Int. Conf. Softw. Eng., Artif. Intell., Netw. Parallel/Distrib. Comput. (SNPD)*, Nov. 2021, pp. 236–241.

[17] Z. Mafi and S.-H. Mirian-Hosseinabadi, "Regression test selection in test-driven development," *Automated Softw. Eng.*, vol. 31, no. 1, p. 9, May 2024.

[18] A. Guldner et al., "Development and evaluation of a reference measurement model for assessing the resource and energy efficiency of software products and components—Green software measurement model (GSMM)," *Future Gener. Comput. Syst.*, vol. 155, pp. 402–418, Jun. 2024.

[19] D. L. T. Rodríguez, V. H. M. García, and G. A. M. Giraldo, "Dynamic model to manage threats in software development projects through artificial intelligence techniques," in *Proc. Workshop Eng. Appl.*, May 2012, pp. 1–6.

[20] H. Sofian, N. A. M. Yunus, and R. Ahmad, "Systematic mapping: Artificial intelligence techniques in software engineering," *IEEE Access*, vol. 10, pp. 51021–51040, 2022.

[21] M. Shehab, L. Abualigah, M. I. Jarrah, O. A. Alomari, and M. S. Daoud, "(AIAM2019) artificial intelligence in software engineering and inverse: Review," *Int. J. Comput. Integr. Manuf.*, vol. 33, nos. 10–11, pp. 1129–1144, Nov. 2020.

[22] B. Kitchenham, R. Pretorius, D. Budgen, O. P. Brereton, M. Turner, M. Niazi, and S. Linkman, "Systematic literature reviews in software engineering—A tertiary study," *Inf. Softw. Technol.*, vol. 52, no. 8, pp. 792–805, 2010.

[23] R. Feldt, F. G. de Oliveira Neto, and R. Torkar, "Ways of applying artificial intelligence in software engineering," in *Proc. IEEE/ACM 6th Int. Workshop Realizing Artif. Intell. Synergies Softw. Eng. (RAISE)*, May 2018, pp. 35–41.

[24] K. H. Shankari and R. Thirumalaiselvi, "A survey on using artificial intelligence techniques in the software development process," *Int. J. Eng. Res. Appl.*, vol. 4, no. 12, pp. 24–33, 2014.

[25] M. Mustaqeem, T. Siddiqui, N. Ahmad Khan, and D. Kumar, "In-depth analysis of various artificial intelligence techniques in software engineering: Experimental study," *J. Inf. Technol. Manage.*, vol. 15, no. 3, pp. 162–181, 2023.

[26] S. Prajapati, B. Prajapati, S. Vegad, and G. Gohil, "Artificial intelligence and software engineering: Status, future trend, and its interaction," *Int. J. Res. Appl. Sci. Eng. Technol.*, vol. 10, no. 3, pp. 1411–1417, Mar. 2022.

[27] S. Saeed and A. Varol, "Software engineering and artificial intelligence: Re-enhancing the lifecycle," in *Proc. Comput. Adv. Trends*, Dec. 2021, pp. 33–44.

[28] 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.

[29] P. Brereton, B. Kitchenham, D. Budgen, and Z. Li, "Using a protocol template for case study planning," in *Proc. Electron. Workshops Comput.*, Jun. 2008, pp. 1–22.

[30] B. Kitchenham, P. Brereton, D. Budgen, M. Turner, J. Bailey, and S. Linkman. (2007). *Protocol for a Tertiary Study of Systematic Literature Reviews and Evidence-Based Guidelines in it and Software Engineering*. [Online]. Available: http://www.durac.uk/ebse/protocol.php

[31] B. Kitchenham, P. Brereton, Z. Li, D. Budgen, and A. Burn, "Repeatability of systematic literature reviews," in *Proc. 15th Annu. Conf. Eval. Assessment Softw. Eng. (EASE)*, Apr. 2011, pp. 46–55.

[32] S. J. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*. London, U.K.: Pearson, 2016.

[33] J. R. Searle, "Minds, brains, and programs," *Behav. Brain Sci.*, vol. 3, no. 3, pp. 417–424, Sep. 1980.

[34] B. Goertzel, "Artificial general intelligence: Concept, state of the art, and future prospects," *J. Artif. Gen. Intell.*, vol. 5, no. 1, pp. 1–48, Dec. 2014.

[35] B. Nick, *Superintelligence: Paths, Dangers, Strategies*, Oxford, U.K.: Oxford Univ. Press, 2014.

[36] D. Poole, A. Mackworth, and R. Goebel, "Computational intelligence: A logical approach," in *Google Scholar Digital Library*, 1998, ch. 1.
[37] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997.
[38] T. Mulgan, "Superintelligence: Paths, dangers, strategies," *Phil. Quart.*, vol. 66, no. 262, pp. 196–203, Jan. 2016, doi: 10.1093/pq/pqv034.
[39] J. McCarthy, "Making robots conscious of their mentaI states," in *Machine Intelligence*, 2000, pp. 3–17.
[40] A. K. Tangirala, *Principles of System Identification: Theory and Practice*. Boca Raton, FL, USA: CRC Press, 2018.
[41] D. Jurafsky, *Speech & Language Processing*. London, U.K.: Pearson, 2023.
[42] D. A. Forsyth and J. Ponce, *Computer Vision: A Modern Approach*. Upper Saddle River, NJ, USA: Prentice-Hall, 2002.
[43] B. Siciliano, O. Khatib, and T. Kröger, *Springer Handbook of Robotics*, vol. 200. Cham, Switzerland: Springer, 2008.
[44] C. M. Bishop and N. M. Nasrabadi, *Pattern Recognition and Machine Learning*, vol. 4. Cham, Switzerland: Springer, 2006.
[45] K. P. Murphy, *Machine Learning: A Probabilistic Perspective*. Cambridge, MA, USA: MIT Press, 2012.
[46] B. Kitchenham, L. Madeyski, D. Budgen, J. Keung, P. Brereton, S. Charters, S. Gibbs, and A. Pohthong, "Robust statistical methods for empirical software engineering," *Empirical Softw. Eng.*, vol. 22, no. 2, pp. 579–630, Apr. 2017.
[47] T. Olsson, M. Ericsson, and A. Wingkvist, "To automatically map source code entities to architectural modules with naive Bayes," *J. Syst. Softw.*, vol. 183, Jan. 2022, Art. no. 111095.
[48] S. Katiyar*, S. Kumar, and H. Walia, "Personality prediction from stack overflow by using Naïve Bayes theorem in data mining," *Int. J. Innov. Technol. Exploring Eng.*, vol. 9, no. 3, pp. 1555–1559, Jan. 2020.
[49] R. Panigrahi, S. K. Kuanar, and L. Kumar, "Application of naive Bayes classifiers for refactoring prediction at the method level," in *Proc. Int. Conf. Comput. Sci., Eng. Appl. (ICCSEA)*, Apr. 2020, pp. 1–6.
[50] Y. Mahmood, N. Kama, A. Azmi, A. S. Khan, and M. Ali, "Software effort estimation accuracy prediction of machine learning techniques: A systematic performance evaluation," *Software: Pract. Exper.*, vol. 52, no. 1, pp. 39–65, Jan. 2022.
[51] F. Palomba, D. A. Tamburri, F. A. Fontana, R. Oliveto, A. Zaidman, and A. Serebrenik, "Beyond technical aspects: How do community smells influence the intensity of code smells?" *IEEE Trans. Softw. Eng.*, vol. 47, no. 1, pp. 108–129, Jan. 2021.
[52] R. Silhavy, P. Silhavy, and Z. Prokopova, "Analysis and selection of a regression model for the use case points method using a stepwise approach," *J. Syst. Softw.*, vol. 125, pp. 1–14, Mar. 2017.
[53] S. Chakraborty, R. Krishna, Y. Ding, and B. Ray, "Deep learning based vulnerability detection: Are we there yet?" *IEEE Trans. Softw. Eng.*, vol. 48, no. 9, pp. 3280–3296, Sep. 2022.
[54] L. Zhao, W. Alhoshan, A. Ferrari, K. J. Letsholo, M. A. Ajagbe, E.-V. Chioasca, and R. T. Batista-Navarro, "Natural language processing for requirements engineering: A systematic mapping study," *ACM Comput. Surveys*, vol. 54, no. 3, pp. 1–41, Apr. 2022.
[55] I. K. Raharjana, D. Siahaan, and C. Fatichah, "User stories and natural language processing: A systematic literature review," *IEEE Access*, vol. 9, pp. 53811–53826, 2021.
[56] T. Chen and M. Li, "The weights can be harmful: Pareto search versus weighted search in multi-objective search-based software engineering," *ACM Trans. Softw. Eng. Methodology*, vol. 32, no. 1, pp. 1–40, Jan. 2023.
[57] S. Singhal, N. Jatana, A. F. Subahi, C. Gupta, O. I. Khalaf, and Y. Alotaibi, "Fault coverage-based test case prioritization and selection using African buffalo optimization," *Comput., Mater. Continua*, vol. 74, no. 3, pp. 6755–6774, 2023.
[58] M. Mafarja, T. Thaher, M. A. Al-Betar, J. Too, M. A. Awadallah, I. Abu Doush, and H. Turabieh, "Classification framework for faulty-software using enhanced exploratory whale optimizer-based feature selection scheme and random forest ensemble learning," *Int. J. Speech Technol.*, vol. 2, pp. 1–43, Feb. 2023.
[59] J. Senanayake, H. Kalutarage, M. O. Al-Kadri, A. Petrovski, and L. Piras, "Android source code vulnerability detection: A systematic literature review," *ACM Comput. Surveys*, vol. 55, no. 9, pp. 1–37, Sep. 2023.
[60] T. H. M. Le, H. Chen, and M. A. Babar, "A survey on data-driven software vulnerability assessment and prioritization," *ACM Comput. Surveys*, vol. 55, no. 5, pp. 1–39, May 2023.
[61] E. A. Almar, A. Peruma, M. W. Mkaouer, C. Newman, A. Ouni, and M. Kessentini, "How we refactor and how we document it? On the use of supervised machine learning algorithms to classify refactoring documentation," *Expert Syst. Appl.*, vol. 167, Apr. 2021, Art. no. 114176.
[62] A. Panichella, "A systematic comparison of search-based approaches for LDA hyperparameter tuning," *Inf. Softw. Technol.*, vol. 130, Feb. 2021, Art. no. 106411.
[63] J. Coelho, M. T. Valente, L. Milen, and L. L. Silva, "Is this GitHub project maintained? Measuring the level of maintenance activity of open-source projects," *Inf. Softw. Technol.*, vol. 122, Jun. 2020, Art. no. 106274.
[64] D. Martens and W. Maalej, "Towards understanding and detecting fake reviews in app stores," *Empirical Softw. Eng.*, vol. 24, no. 6, pp. 3316–3355, Dec. 2019.
[65] R. Pan, M. Bagherzadeh, T. A. Ghaleb, and L. Briand, "Test case selection and prioritization using machine learning: A systematic literature review," *Empirical Softw. Eng.*, vol. 27, no. 2, p. 29, Mar. 2022.
[66] S. Wattanakriengkrai, P. Thongtanunam, C. Tantithamthavorn, H. Hata, and K. Matsumoto, "Predicting defective lines using a model-agnostic technique," *IEEE Trans. Softw. Eng.*, vol. 48, no. 5, pp. 1480–1496, May 2022.
[67] T. Shippey, D. Bowes, and T. Hall, "Automatically identifying code features for software defect prediction: Using AST N-grams," *Inf. Softw. Technol.*, vol. 106, pp. 142–160, Feb. 2019.
[68] R. Heradio, D. Fernandez-Amoros, J. A. Galindo, D. Benavides, and D. Batory, "Uniform and scalable sampling of highly configurable systems," *Empirical Softw. Eng.*, vol. 27, no. 2, p. 44, Mar. 2022.
[69] J. Campos, Y. Ge, N. Albunian, G. Fraser, M. Eler, and A. Arcuri, "An empirical evaluation of evolutionary algorithms for unit test suite generation," *Inf. Softw. Technol.*, vol. 104, pp. 207–235, Dec. 2018.
[70] J. M. Rojas, G. Fraser, and A. Arcuri, "Seeding strategies in search-based unit test generation," *Softw. Test., Verification Rel.*, vol. 26, no. 5, pp. 366–401, Aug. 2016.
[71] C. Hettiarachchi, H. Do, and B. Choi, "Risk-based test case prioritization using a fuzzy expert system," *Inf. Softw. Technol.*, vol. 69, pp. 1–15, Jan. 2016.
[72] Y. Yuan and W. Banzhaf, "ARJA: Automated repair of Java programs via multi-objective genetic programming," *IEEE Trans. Softw. Eng.*, vol. 46, no. 10, pp. 1040–1067, Oct. 2020.
[73] C. Henard, M. Papadakis, G. Perrouin, J. Klein, P. Heymans, and Y. Le Traon, "Bypassing the combinatorial explosion: Using similarity to generate and prioritize T-Wise test configurations for software product lines," *IEEE Trans. Softw. Eng.*, vol. 40, no. 7, pp. 650–670, Jul. 2014.
[74] T. Lewowski and L. Madeyski, "How far are we from reproducible research on code smell detection? A systematic literature review," *Inf. Softw. Technol.*, vol. 144, Apr. 2022, Art. no. 106783.
[75] V. Nair, Z. Yu, T. Menzies, N. Siegmund, and S. Apel, "Finding faster configurations using FLASH," *IEEE Trans. Softw. Eng.*, vol. 46, no. 7, pp. 794–811, Jul. 2020.
[76] J. Nam, W. Fu, S. Kim, T. Menzies, and L. Tan, "Heterogeneous defect prediction," *IEEE Trans. Softw. Eng.*, vol. 44, no. 9, pp. 874–896, Sep. 2018.
[77] 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.
[78] J. Xuan, H. Jiang, Y. Hu, Z. Ren, W. Zou, Z. Luo, and X. Wu, "Towards effective bug triage with software data reduction techniques," *IEEE Trans. Knowl. Data Eng.*, vol. 27, no. 1, pp. 264–280, Jan. 2015.
[79] Y. Tian, D. Lo, X. Xia, and C. Sun, "Automated prediction of bug report priority using multi-factor analysis," *Empirical Softw. Eng.*, vol. 20, no. 5, pp. 1354–1383, Oct. 2015.
[80] S. Combéfis, "Automated code assessment for education: Review, classification and perspectives on techniques and tools," *Software*, vol. 1, no. 1, pp. 3–30, Feb. 2022.
[81] K. Moran, C. Bernal-Cárdenas, M. Curcio, R. Bonett, and D. Poshyvanyk, "Machine learning-based prototyping of graphical user interfaces for mobile apps," *IEEE Trans. Softw. Eng.*, vol. 46, no. 2, pp. 196–221, Feb. 2020.
[82] F. Peters, T. Menzies, L. Gong, and H. Zhang, "Balancing privacy and utility in cross-company defect prediction," *IEEE Trans. Softw. Eng.*, vol. 39, no. 8, pp. 1054–1068, Aug. 2013.
[83] M. Elallaoui, K. Nafil, and R. Touahni, "Automatic transformation of user stories into UML use case diagrams using NLP techniques," *Proc. Comput. Sci.*, vol. 130, pp. 42–49, Apr. 2018.
[84] G. Lucassen, F. Dalpiaz, J. M. E. M. van der Werf, and S. Brinkkemper, "Improving agile requirements: The quality user story framework and tool," *Requirements Eng.*, vol. 21, no. 3, pp. 383–403, Sep. 2016.

[85] L. Nie, H. Jiang, Z. Ren, Z. Sun, and X. Li, "Query expansion based on crowd knowledge for code search," *IEEE Trans. Services Comput.*, vol. 9, no. 5, pp. 771–783, Sep. 2016.
[86] C. Treude, M. P. Robillard, and B. Dagenais, "Extracting development tasks to navigate software documentation," *IEEE Trans. Softw. Eng.*, vol. 41, no. 6, pp. 565–581, Jun. 2015.
[87] G. Bavota, R. Oliveto, M. Gethers, D. Poshyvanyk, and A. De Lucia, "Methodbook: Recommending move method refactorings via relational topic models," *IEEE Trans. Softw. Eng.*, vol. 40, no. 7, pp. 671–694, Jul. 2014.
[88] X.-N. Shen, L. L. Minku, N. Marturi, Y.-N. Guo, and Y. Han, "A Q-learning-based memetic algorithm for multi-objective dynamic software project scheduling," *Inf. Sci.*, vol. 428, pp. 1–29, Feb. 2018.
[89] W. Fu, T. Menzies, and X. Shen, "Tuning for software analytics: Is it really necessary?" *Inf. Softw. Technol.*, vol. 76, pp. 135–146, Aug. 2016.
[90] Y. Xue, J. Zhong, T. H. Tan, Y. Liu, W. Cai, M. Chen, and J. Sun, "IBED: Combining IBEA and DE for optimal feature selection in software product line engineering," *Appl. Soft Comput.*, vol. 49, pp. 1215–1231, Dec. 2016.
[91] J. Petke, M. B. Cohen, M. Harman, and S. Yoo, "Practical combinatorial interaction testing: Empirical findings on efficiency and early fault detection," *IEEE Trans. Softw. Eng.*, vol. 41, no. 9, pp. 901–924, Sep. 2015.
[92] B. S. Ahmed, T. S. Abdulsamad, and M. Y. Potrus, "Achievement of minimized combinatorial test suite for configuration-aware software functional testing using the cuckoo search algorithm," *Inf. Softw. Technol.*, vol. 66, pp. 13–29, Oct. 2015.
[93] W. Mkaouer, M. Kessentini, A. Shaout, P. Koligheu, S. Bechikh, K. Deb, and A. Ouni, "Many-objective software remodularization using NSGA-III," *ACM Trans. Softw. Eng. Methodology*, vol. 24, no. 3, pp. 1–45, May 2015.
[94] G. Fraser, A. Arcuri, and P. Mcminn, "A memetic algorithm for whole test suite generation," *J. Syst. Softw.*, vol. 103, pp. 311–327, May 2015.
[95] L. Tran-Thanh, S. Stein, A. Rogers, and N. R. Jennings, "Efficient crowdsourcing of unknown experts using bounded multi-armed bandits," *Artif. Intell.*, vol. 214, pp. 89–111, Sep. 2014.
[96] G. Fraser and A. Arcuri, "Whole test suite generation," *IEEE Trans. Softw. Eng.*, vol. 39, no. 2, pp. 276–291, Feb. 2013.
[97] T. Berger, S. She, R. Lotufo, A. Wasowski, and K. Czarnecki, "A study of variability models and languages in the systems software domain," *IEEE Trans. Softw. Eng.*, vol. 39, no. 12, pp. 1611–1640, Dec. 2013.
[98] G. Fraser and A. Arcuri, "1600 faults in 100 projects: Automatically finding faults while achieving high coverage with EvoSuite," *Empirical Softw. Eng.*, vol. 20, no. 3, pp. 611–639, Jun. 2015.
[99] J. del Sagrado, I. M. del Águila, and F. J. Orellana, "Multi-objective ant colony optimization for requirements selection," *Empirical Softw. Eng.*, vol. 20, no. 3, pp. 577–610, Jun. 2015.
[100] H. Mao, Y. Chen, M. Jaeger, T. D. Nielsen, K. G. Larsen, and B. Nielsen, "Learning deterministic probabilistic automata from a model checking perspective," *Mach. Learn.*, vol. 105, no. 2, pp. 255–299, Nov. 2016.
[101] M. Paasivaara and C. Lassenius, "Challenges and success factors for large-scale agile transformations," in *Proc. Sci. Workshop Proc.*, May 2016, pp. 1–5.
[102] G. Bavota, M. Linares-Vásquez, C. E. Bernal-Cárdenas, M. D. Penta, R. Oliveto, and D. Poshyvanyk, "The impact of API Change- and fault-proneness on the user ratings of Android apps," *IEEE Trans. Softw. Eng.*, vol. 41, no. 4, pp. 384–407, Apr. 2015.
[103] C. Tantithamthavorn, A. E. Hassan, and K. Matsumoto, "The impact of class rebalancing techniques on the performance and interpretation of defect prediction models," *IEEE Trans. Softw. Eng.*, vol. 46, no. 11, pp. 1200–1219, Nov. 2020.
[104] M. Aniche, E. Maziero, R. Durelli, and V. H. S. Durelli, "The effectiveness of supervised machine learning algorithms in predicting software refactoring," *IEEE Trans. Softw. Eng.*, vol. 48, no. 4, pp. 1432–1450, Apr. 2022.
[105] F. Pecorelli, D. Di Nucci, C. De Roover, and A. De Lucia, "A large empirical assessment of the role of data balancing in machine-learning-based code smell detection," *J. Syst. Softw.*, vol. 169, Nov. 2020, Art. no. 110693.
[106] T. Hall, M. Zhang, D. Bowes, and Y. Sun, "Some code smells have a significant but small effect on faults," *ACM Trans. Softw. Eng. Methodol.*, vol. 23, no. 4, pp. 1–39, Sep. 2014.
[107] L. L. Minku and X. Yao, "Ensembles and locality: Insight on improving software effort estimation," *Inf. Softw. Technol.*, vol. 55, no. 8, pp. 1512–1528, Aug. 2013.
[108] S. Gao, C. Gao, Y. He, J. Zeng, L. Nie, X. Xia, and M. Lyu, "Code structure-guided transformer for source code summarization," *ACM Trans. Softw. Eng. Methodol.*, vol. 32, no. 1, pp. 1–32, Jan. 2023.
[109] G. Giray, K. E. Bennin, Ö. Köksal, Ö. Babur, and B. Tekinerdogan, "On the use of deep learning in software defect prediction," *J. Syst. Softw.*, vol. 195, Jan. 2023, Art. no. 111537.
[110] Z. Gao, X. Xia, D. Lo, J. Grundy, X. Zhang, and Z. Xing, "I know what you are searching for: Code snippet recommendation from stack overflow posts," *ACM Trans. Softw. Eng. Methodol.*, vol. 32, no. 3, pp. 1–42, Jul. 2023.
[111] M. R. I. Rabin, N. D. Q. Bui, K. Wang, Y. Yu, L. Jiang, and M. A. Alipour, "On the generalizability of neural program models with respect to semantic-preserving program transformations," *Inf. Softw. Technol.*, vol. 135, Jul. 2021, Art. no. 106552.
[112] W. Wang, Y. Zhang, Y. Sui, Y. Wan, Z. Zhao, J. Wu, P. S. Yu, and G. Xu, "Reinforcement-Learning-Guided source code summarization using hierarchical attention," *IEEE Trans. Softw. Eng.*, vol. 48, no. 1, pp. 102–119, Jan. 2022.
[113] R. Yedida and T. Menzies, "On the value of oversampling for deep learning in software defect prediction," *IEEE Trans. Softw. Eng.*, vol. 48, no. 8, pp. 3103–3116, Aug. 2022.
[114] W. Hua, Y. Sui, Y. Wan, G. Liu, and G. Xu, "FCCA: Hybrid code representation for functional clone detection using attention networks," *IEEE Trans. Rel.*, vol. 70, no. 1, pp. 304–318, Mar. 2021.
[115] M. M. Rahman, Y. Watanobe, and K. Nakamura, "Source code assessment and classification based on estimated error probability using attentive LSTM language model and its application in programming education," *Appl. Sci.*, vol. 10, no. 8, p. 2973, Apr. 2020.
[116] F. Ullah, H. Naeem, S. Jabbar, S. Khalid, M. A. Latif, F. Al-turjman, and L. Mostarda, "Cyber security threats detection in Internet of Things using deep learning approach," *IEEE Access*, vol. 7, pp. 124379–124389, 2019.
[117] X. Ren, Z. Xing, X. Xia, D. Lo, X. Wang, and J. Grundy, "Neural network-based detection of self-admitted technical debt," *ACM Trans. Softw. Eng. Methodol.*, vol. 28, no. 3, pp. 1–45, Jul. 2019.
[118] Y. Li, S. Wang, T. N. Nguyen, and S. Van Nguyen, "Improving bug detection via context-based code representation learning and attention-based neural networks," *Proc. ACM Program. Lang.*, vol. 3, no. 2, pp. 1–30, Oct. 2019.
[119] G. Fan, X. Diao, H. Yu, K. Yang, and L. Chen, "Software defect prediction via attention-based recurrent neural network," *Sci. Program.*, vol. 2019, pp. 1–14, Apr. 2019.
[120] 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.
[121] S. Jha, R. Kumar, L. Hoang Son, M. Abdel-Basset, I. Priyadarshini, R. Sharma, and H. Viet Long, "Deep learning approach for software maintainability metrics prediction," *IEEE Access*, vol. 7, pp. 61840–61855, 2019.
[122] 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.
[123] C. M. Nguyen, R. Sebastiani, P. Giorgini, and J. Mylopoulos, "Multi-objective reasoning with constrained goal models," *Requirements Eng.*, vol. 23, no. 2, pp. 189–225, Jun. 2018.
[124] A. Jovanovic, D. Lime, and O. H. Roux, "Integer parameter synthesis for real-time systems," *IEEE Trans. Softw. Eng.*, vol. 41, no. 5, pp. 445–461, May 2015.
[125] D. Tomar and S. Agarwal, "Prediction of defective software modules using class imbalance learning," *Appl. Comput. Intell. Soft Comput.*, vol. 2016, pp. 1–12, May 2016.
[126] M. Jorayeva, A. Akbulut, C. Catal, and A. Mishra, "Machine learning-based software defect prediction for mobile applications: A systematic literature review," *Sensors*, vol. 22, no. 7, p. 2551, Mar. 2022.
[127] F. Matloob, T. M. Ghazal, N. Taleb, S. Aftab, M. Ahmad, M. A. Khan, S. Abbas, and T. R. Soomro, "Software defect prediction using ensemble learning: A systematic literature review," *IEEE Access*, vol. 9, pp. 98754–98771, 2021.
[128] N. Limsettho, K. E. Bennin, J. W. Keung, H. Hata, and K. Matsumoto, "Cross project defect prediction using class distribution estimation and oversampling," *Inf. Softw. Technol.*, vol. 100, pp. 87–102, Aug. 2018.

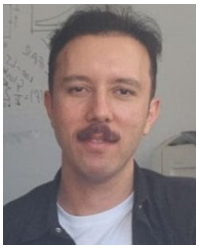
[129] D. Bowes, T. Hall, and J. Petrić, "Software defect prediction: Do different classifiers find the same defects?" *Softw. Quality J.*, vol. 26, no. 2, pp. 525–552, Jun. 2018.
[130] L. K. Shar, L. C. Briand, and H. B. K. Tan, "Web application vulnerability prediction using hybrid program analysis and machine learning," *IEEE Trans. Dependable Secure Comput.*, vol. 12, no. 6, pp. 688–707, Nov. 2015.
[131] P. He, B. Li, X. Liu, J. Chen, and Y. Ma, "An empirical study on software defect prediction with a simplified metric set," *Inf. Softw. Technol.*, vol. 59, pp. 170–190, Mar. 2015.
[132] F. Palomba, G. Bavota, M. D. Penta, R. Oliveto, D. Poshyvanyk, and A. De Lucia, "Mining version histories for detecting code smells," *IEEE Trans. Softw. Eng.*, vol. 41, no. 5, pp. 462–489, May 2015.
[133] R. Scandariato, J. Walden, A. Hovsepyan, and W. Joosen, "Predicting vulnerable software components via text mining," *IEEE Trans. Softw. Eng.*, vol. 40, no. 10, pp. 993–1006, Oct. 2014.
[134] L. Madeyski and M. Jureczko, "Which process metrics can significantly improve defect prediction models? An empirical study," *Softw. Quality J.*, vol. 23, no. 3, pp. 393–422, Sep. 2015.
[135] M. Shepperd, D. Bowes, and T. Hall, "Researcher bias: The use of machine learning in software defect prediction," *IEEE Trans. Softw. Eng.*, vol. 40, no. 6, pp. 603–616, Jun. 2014.
[136] S. Huda, K. Liu, M. Abdelrazek, A. Ibrahim, S. Alyahya, H. Al-Dossari, and S. Ahmad, "An ensemble oversampling model for class imbalance problem in software defect prediction," *IEEE Access*, vol. 6, pp. 24184–24195, 2018.
[137] E. Soremekun, S. Udeshi, and S. Chattopadhyay, "Astraea: Grammar-based fairness testing," *IEEE Trans. Softw. Eng.*, vol. 48, no. 12, pp. 5188–5211, Dec. 2022.
[138] N. Ali, Y.-G. Guéhéneuc, and G. Antoniol, "Trustrace: Mining software repositories to improve the accuracy of requirement traceability links," *IEEE Trans. Softw. Eng.*, vol. 39, no. 5, pp. 725–741, May 2013.
[139] S. Shivaji, E. J. Whitehead, R. Akella, and S. Kim, "Reducing features to improve code change-based bug prediction," *IEEE Trans. Softw. Eng.*, vol. 39, no. 4, pp. 552–569, Apr. 2013.
[140] J. Fischbach, J. Frattini, A. Vogelsang, D. Mendez, M. Unterkalmsteiner, A. Wehrle, P. R. Henao, P. Yousefi, T. Juricic, J. Radduenz, and C. Wiecher, "Automatic creation of acceptance tests by extracting conditionals from requirements: NLP approach and case study," *J. Syst. Softw.*, vol. 197, Mar. 2023, Art. no. 111549.
[141] B. Lin, N. Cassee, A. Serebrenik, G. Bavota, N. Novielli, and M. Lanza, "Opinion mining for software development: A systematic literature review," *ACM Trans. Softw. Eng. Methodology*, vol. 31, no. 3, pp. 1–41, Mar. 2022, doi: 10.1145/3490388.
[142] L. Zhao, W. Alhoshan, A. Ferrari, K. J. Letsholo, M. A. Ajagbe, E.-V. Chioasca, and R. T. Batista-Navarro, "Natural language processing for requirements engineering," *ACM Comput. Surveys*, vol. 54, no. 3, pp. 1–41, Apr. 2022.
[143] R. Jongeling, P. Sarkar, S. Datta, and A. Serebrenik, "On negative results when using sentiment analysis tools for software engineering research," *Empirical Softw. Eng.*, vol. 22, no. 5, pp. 2543–2584, Oct. 2017.
[144] H. Femmer, D. Méndez Fernández, S. Wagner, and S. Eder, "Rapid quality assurance with requirements smells," *J. Syst. Softw.*, vol. 123, pp. 190–213, Jan. 2017.
[145] A. Di Sorbo, F. Zampetti, A. Visaggio, M. Di Penta, and S. Panichella, "Automated identification and qualitative characterization of safety concerns reported in UAV software platforms," *ACM Trans. Softw. Eng. Methodol.*, vol. 32, no. 3, pp. 1–37, Apr. 2023, doi: 10.1145/3564821.
[146] M. Izadi, K. Akbari, and A. Heydarnoori, "Predicting the objective and priority of issue reports in software repositories," *Empirical Softw. Eng.*, vol. 27, no. 2, p. 50, Mar. 2022.
[147] S. Ali, Y. Hafeez, N. Z. Jhanjhi, M. Humayun, M. Imran, A. Nayyar, S. Singh, and I.-H. Ra, "Towards pattern-based change verification framework for cloud-enabled healthcare component-based," *IEEE Access*, vol. 8, pp. 148007–148020, 2020.
[148] F. Calefato, F. Lanubile, F. Maiorano, and N. Novielli, "Sentiment polarity detection for software development," in *Proc. 40th Int. Conf. Softw. Eng.*, May 2018, p. 128.
[149] Q. Huang, E. Shihab, X. Xia, D. Lo, and S. Li, "Identifying self-admitted technical debt in open source projects using text mining," *Empirical Softw. Eng.*, vol. 23, no. 1, pp. 418–451, Feb. 2018.
[150] T. Chen and M. Li, "Do performance aspirations matter for guiding software configuration tuning? An empirical investigation under dual performance objectives," *ACM Trans. Softw. Eng. Methodology*, vol. 32, no. 3, pp. 1–41, Jul. 2023.
[151] Y. Sui, X. Cheng, G. Zhang, and H. Wang, "Flow2Vec: Value-flow-based precise code embedding," *Proc. ACM Program. Lang.*, vol. 4, no. 4, pp. 1–27, Nov. 2020.
[152] R. R. Sahoo and M. Ray, "PSO based test case generation for critical path using improved combined fitness function," *J. King Saud Univ. Comput. Inf. Sci.*, vol. 32, no. 4, pp. 479–490, May 2020.
[153] R. Calinescu, D. Weyns, S. Gerasimou, M. U. Iftikhar, I. Habli, and T. Kelly, "Engineering trustworthy self-adaptive software with dynamic assurance cases," *IEEE Trans. Softw. Eng.*, vol. 44, no. 11, pp. 1039–1069, Nov. 2018.
[154] M. Martinez and M. Monperrus, "Mining software repair models for reasoning on the search space of automated program fixing," *Empirical Softw. Eng.*, vol. 20, no. 1, pp. 176–205, Feb. 2015.
[155] S. Kassaymeh, S. Abdullah, M. A. Al-Betar, and M. Alweshah, "Salp swarm optimizer for modeling the software fault prediction problem," *J. King Saud Univ. Comput. Inf. Sci.*, vol. 34, no. 6, pp. 3365–3378, Jun. 2022.
[156] L. Pascarella, F. Palomba, and A. Bacchelli, "Fine-grained just-in-time defect prediction," *J. Syst. Softw.*, vol. 150, pp. 22–36, Apr. 2019.
[157] S. Luan, D. Yang, C. Barnaby, K. Sen, and S. Chandra, "Aroma: Code recommendation via structural code search," *Proc. ACM Program. Lang.*, vol. 3, no. 5, pp. 1–28, Oct. 2019.
[158] X. Xia, D. Lo, X. Wang, and X. Yang, "Collective personalized change classification with multiobjective search," *IEEE Trans. Rel.*, vol. 65, no. 4, pp. 1810–1829, Dec. 2016.
[159] M. Weyssow, H. Sahraoui, and E. Syriani, "Recommending metamodel concepts during modeling activities with pre-trained language models," *Softw. Syst. Model.*, vol. 21, no. 3, pp. 1071–1089, Jun. 2022.

USMAN KHAN DURRANI (Member, IEEE) received the Ph.D. degree in information systems from RMIT University, Australia. He has over 20 years of academic and IT industry experience. With a strong foundation in software engineering, AI, and data analytics, he has extensive industry exposure at prestigious organizations, such as IBM, ANZ, and Fujitsu in various technical and managerial roles. He is a Distinguished Academic and Seasoned Researcher. His publication record and active engagement in scientific societies underscore his commitment to bridging academia and industry, enriching the academic landscape with practical insights and cutting-edge research. His research interests include adaptable software engineering processes, application of AI for lean engineering, lean project management, and the integration of gamification and emerging technologies in education.

MUSTAFA AKPINAR (Member, IEEE) was born in Sakarya, Türkiye, in 1983. He received the B.S. degree in mechanical engineering and the B.S. degree in computer engineering, in 2006 and 2010, respectively, the M.S. degree in mechanical engineering and the M.S. degree in computer engineering, in 2010 and 2014, respectively, and the Ph.D. degree in computer engineering from Sakarya University, Türkiye, in 2017. From 2006 to 2012, he worked in the manufacturing and energy sectors as an Engineer with several projects about demand forecasting, sheet metal forming, and optimization. From 2012 to 2018, he was a Research Assistant with the Computer Engineering Department, Sakarya University. After the Ph.D. degree, he started as an Assistant Professor with the Software Engineering Department, Sakarya University. Since 2022, he has been an Assistant Professor with the Computer and Information Science Department, Higher Colleges of Technology, United Arab Emirates. His research interests include statistics, time series analysis, demand forecasting, finite-element methods, sheet metal forming, CAD/CAE, optimization algorithms, and machine learning. He is the Secretary of the Education Society in the IEEE UAE Section.

MUHAMMED FATIH ADAK (Member, IEEE) received the master's degree in computer engineering from Kocaeli University, Kocaeli, Turkey, in 2012, and the Ph.D. degree in computer engineering from Sakarya University, Turkey, in 2016. From 2012 to 2017, he served as a Research Assistant at Sakarya University, where he is currently an Assistant Professor. He has contributed to various research projects and authored course books. His research interests include machine learning, software testing, smart technologies, bioinformatics, and the classification of data from electronic noses and sensors.

ABDULLAH TALHA KABAKUS received the bachelor's degree in computer engineering from Çankaya University, in 2010, the master's degree in computer engineering from Gazi University, in 2014, and the Philosophy of Doctorate degree in electrical-electronics and computer engineering from Duzce University, in 2017. He is currently an Associate Professor with the Department of Computer Engineering, Faculty of Engineering, Duzce University. His research interests include mobile security, deep learning, and natural language processing.



MUHAMMED MARUF ÖZTÜRK received the bachelor's degree in computer engineering from Pamukkale University and the Ph.D. degree in computer engineering from Sakarya University. He is currently an Associate Professor with the Department of Computer Engineering, Faculty of Engineering, Suleyman Demirel University. His research interests include encompass artificial intelligence in software testing, energy-efficient software systems, software effort estimation, and hyperparameter optimization.

MOHAMMED SALEH (Member, IEEE) received the bachelor's degree in electrical engineering (computer major) from the University of North Carolina at Charlotte, USA, and the Post Graduate Diploma degree in education and training and the Master of Engineering and Ph.D. degrees in computer/telecommunication engineering from Victoria University, Australia. He has over 25 years of experience in education and research at institutions in Australia and Middle East. He has mentored several student startup projects in various courses throughout his academic career. He recently participated with HCT students in the UN Framework Convention on Climate Change's (UNFCCC) Middle East and North Africa Climate Week 2022. His fields of research and topics of collaborative projects cover autonomous systems and transport, smart cities, smart homes, smart grids, and unmanned autonomous vehicles. He is the Vice Chair of the Education Society at the IEEE UAE Section.