Review of ways to apply machine learning methods in software engineering

Jameleh Asaad* and *Elena* Avksentieva

ITMO University, Saint Petersburg, Russia

Abstract. This article reviews the integration of machine learning (ML) techniques into Software Engineering (SE) across various phases of the software development life cycle (SDLC). The purpose is to investigate the applications of ML in SE, analyze its methodologies, present findings, and draw conclusions regarding its impact. The study categorized ML applications in SE and assessed the performance of various ML algorithms. Authors identified ML applications in SDLC phases, including requirements analysis, design, implementation, testing, and maintenance. ML algorithms, such as supervised and unsupervised learning, are employed for tasks like software requirement identification, design pattern recognition, code generation, and automated testing. In summary, we find that ML-based techniques are experiencing a substantial surge in adoption within the field of software engineering. Nevertheless, it is evident that substantial endeavors are needed to establish thorough comparisons and synergies among these approaches, perform meaningful evaluations grounded in detailed real-world implementations that are applicable to industrial software development. Therefore, our key takeaway is the necessity for a shift in focus towards reproducible research, prioritizing this over isolated novel concepts. Failure to do so may result in the limited practical implementation of these promising applications.

1 Introduction

In contemporary landscape, the spreading of discussions and scholarly discourse surrounding machine learning (ML), data mining, big data analytics, and artificial intelligence (AI) is readily evident. These domains have become integral components of scientific dialogue, reflecting their substantial influence on modern society. The intensifying interest in these fields has been further accentuated by the emergence of ChatGPT in November 2022, representing a watershed moment in AI development. This development highlights the escalating importance of these technologies and their ongoing transformative impact on various aspects of society, particularly in the realms of technology and information processing.

Machine learning deals with the issue of how to build programs that improve their performance through experience. Machine learning algorithms have proven to be of great practical value in a variety of application domains. Machine learning has been successfully applied in many areas of software engineering, ranging from features extraction to testing to

* Corresponding author: jamelehasaad@gmail.com

bug fixing. If software developers had a better grasp of machine learning approaches, their assumptions, and guarantees, they might adopt and select the best techniques for their intended applications. To meet the needs of changing approaches to software development, future software engineering (SE) techniques and tools will need to be much more automated, lightweight, adaptable, and scalable to keep pace with increased developer productivity. The growing dependency on applications incorporating machine learning (ML) components necessitates the establishment of advanced engineering methodologies to guarantee their development with robustness and future-proof capabilities. This escalating reliance underscores the critical need for the application of mature and well-structured engineering techniques.

Furthermore, software is an indispensable component of the majority of systems and is integrated into the daily lives of society. With the advancement of technologies such as open systems and highly automated or networked devices, software systems are becoming very complex [1]. Additionally, several people from different areas of expertise are usually required to be involved in a software project, which also increases its complexity. Since software is developed by humans, it is usual that people make mistakes; thus, in every commercial piece of software, some errors always occur [2], and as the level of complexity increases, these errors become more significant [3]. Automating the SDLC process using machine learning can help solve these problems. We have analyzed several successful examples of the effective use of machine learning techniques in software development.

2 The research question and methodology

The primary objective of this research study is to investigate the utilization of machine learning (ML) methodologies within the software development life cycle (SDLC) and assess their overall performance. By undertaking this comprehensive examination, our aim is to shed light on the current landscape of ML applications in software engineering and to pinpoint areas where enhancements are essential to optimize the efficacy of these methods. To attain this goal, we have formulated a set of research questions that serve as guiding pillars throughout this inquiry:

Research Question 1 (RQ1): What categories of software applications have been identified or documented within the present phase of software development?

Research Question 2 (RQ2): Which specific ML algorithms have been employed during this phase of software development?

Research Question 3 (RQ3): What is the performance evaluation of ML-based techniques, and how do they compare with their non-ML-based counterparts in terms of effectiveness and efficiency?

Structuring the literature review involved breaking down the overall task into several smaller steps so as to enable us to explore the literature for systematically extract relevant information. Firstly, key search strings were utilized: ‘Machine learning for software engineering’, ‘Machine learning + software engineering’, ‘Machine learning for SDLC’, ‘Machine learning + (and | for | +) + software requirement’, ‘Machine learning + (and | for | +) + software design’, ‘Machine learning + (and | for | +) + software testing’, ‘Machine learning + (and | for | +) + software construction’ and ‘Machine learning + (and | for | +) + software maintenance’ so as to identify a baseline set of research papers. Google, Google Scholar, and digital libraries of publications from ACM and IEEE were used to find these publications.

Following the completion of the initial phase of the literature review, the shortlists for each search term were further evaluated. The relevance of publications was examined by reading each abstract and conclusion. Each publication was sorted according to the number of citations it had and the year it was published. The next step of the process involved reading the publications in detail and making further evaluations in relation to their relevance. Overall, the results of this systematic approach are present in Section 3.

3 Background and related works

The interaction between software engineering (SE) and machine learning (ML) has been studied by researchers for a long time [1-3]. The first Symposium on Software Engineering for Machine Learning Applications (SEMLA) at Polytechnique Montréal was organized on 12 and 13 June 2018, with the support of Polytechnique Montréal's Department of Computer Engineering and Software Engineering, the Institute for Data Valorization (IVADO), SAP, and Red Hat. Around 160 participants from 160 different countries attended the event, including students, professors, and professionals from the business sector.

On the contrary, some studies highlight the gap between the SE (Software Engineering) and ML (Machine Learning) communities, attributing it, in part, to their differing focuses. The ML community primarily concerns itself with algorithms and their performance, whereas the SE community is dedicated to developing and deploying software-intensive systems [4].

However, there are areas of synergy when these two communities collaborate. "SE for ML" involves SE experts taking on responsibilities related to engineering ML systems, encompassing tasks like designing, creating, and maintaining software systems that support ML. Researchers in this field strive to identify distinctions between ML system design and conventional software, aiming to develop new strategies and tools to bridge these disparities.

Conversely, "ML for SE" entails the adaptation of AI technologies to address various SE tasks, including software fault prediction, code smell detection, reusability metrics prediction, and cost estimation. Researchers leverage ML models derived from SE data, such as source code, requirement specifications, and test cases, to enhance software engineering's efficiency and effectiveness.

Machine learning (ML) is a branch of research that offers computers the capacity to learn without being explicitly programmed. It was first described by Arthur Samuel in 1959.

The term "software engineering," coined by David Parnas in 1972, is officially defined by the IEEE as the "application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software," essentially applying engineering principles to software creation. This discipline revolves around employing structured methodologies to execute the necessary procedures for analyzing, designing, implementing, and maintaining information systems. Effective planning and preparation are vital in software projects to ensure timely delivery and the production of high-quality software.

Central to software development is the Software Development Life Cycle (SDLC), which constitutes the complete process of constructing any software product. SDLC encompasses a variety of methodologies, including Agile, Waterfall, DevOps, V-Model, Iterative, Dynamic System Development Model, Extreme Programming, Feature Driven Development, Joint Application Development, Spiral, Rapid Application Development, and Lean practices.

The SDLC typically comprises several phases: requirements analysis, design, implementation, testing, and maintenance. These stages collectively govern the journey of software development, ensuring its successful execution and continued functionality [5].

At the present time, software engineering has transitioned from traditional waterfall models to agile software development. A waterfall model is a sequential process where the success of each stage depends on the success of the previous stages. All requirements are thought to be clearly established at project inception and essentially stable after that. Agile processes are iterative software development techniques that offer adaptability and flexibility in response to changing conditions while producing high-quality software. It emphasizes informal, adaptable project management that will improve communication and transparency.

This paper focuses on machine learning for software engineering by systematically reviewing the machine learning literature for software development tasks.

4 Machine Learning in Software Engineering

As previously noted, the Software Development Life Cycle (SDLC) encompasses five distinct phases [5]. In this section, we will delve into the research inquiries associated with each of these SDLC phases.

4.1 ML for requirement engineering

Throughout the software development process, requirement engineering (RE) is essential. Prioritization and requirement identification are the key stages of the RE process [6].

RQ1: ML-based approaches were employed to identify distinct software requirements, encompassing functional requirements (FRs) [7-11] and non-functional requirements (NFRs) [12-18].

RQ2: While numerous machine learning algorithms and techniques are accessible for text processing, they can generally be categorized into two primary groups: supervised learning algorithms (SL) and unsupervised learning algorithms (USL). Additionally, there exists a hybrid category known as semi-supervised learning (SSL), bridging the gap between supervised and unsupervised approaches.

The results of this investigation highlighted two predominant categories of machine learning algorithms within the reviewed primary research. These algorithms primarily belong to two classes: supervised learning (SL) and unsupervised learning (USL). Notably, some primary studies employed thematic analysis or qualitative coding methods. Furthermore, the selected studies illustrated that USL algorithms, particularly Latent Dirichlet Allocation (LDA), emerged as the favored type of machine learning algorithm. Support Vector Machine (SVM) ranked as the second-most prevalent ML algorithm category. It was intriguing to note that a subset of the chosen primary studies eschewed algorithmic approaches, opting instead for thematic or qualitative coding techniques to discern and categorize various software requirements.

The process can be distilled into three primary phases: text preprocessing to eliminate irrelevant content, the application of various ML algorithms during the learning stage, and the subsequent analysis and evaluation of the methodology used by these algorithms.

The chosen research articles revealed a total of six alternative Natural Language Processing (NLP) preprocessing methods. The following is a quick explanation of the many preprocessing methods found.

Text preprocessing involves several key steps, including the removal of stop words, which are auxiliary verbs like "be," "do," and "have," as well as articles such as "the," "a," and "an" [19]. Tokenization, another essential technique, breaks down text into individual words [20]. Additionally, case unification standardizes text to lowercase or uppercase, while stemming reduces words to their base form, such as "goes," "gone," and "going" becoming "go" [19]. Punctuation removal eliminates various punctuations like commas, semicolons, question marks, and exclamation marks. Notably, some studies lack reporting on the majority of these preprocessing steps. In many cases, machine learning-based techniques are considered 'black boxes,' with limited insight into their inner workings.

Regarding performance evaluation, not all selected studies conducted thorough assessments. While LDA and SVM were applied in various research articles, it's noteworthy that their performance outcomes exhibited significant variations. For instance, the LDA algorithm demonstrated strong performance in one study [16], but its effectiveness was less impressive in another [17].

4.2 Software design

In the software development life cycle, it is the most inventive phase. This phase's objective is to arrange or plan the required definition. It is the planning and issue-solving process for a software solution. It involves software designers and developers specifying the strategy for a fix. This phase results in a software design document (SDD).

Software design is a highly complex and challenging activity. Nevertheless, using software design patterns makes this phase more organized. A software design pattern can be defined as a presupposed structure of classes organized and interacting in a particular manner to solve a recurring design problem.

RQ1: The studies show that ML is able to be used to avoid some problems in this phase, for instance, detection of the bad smells earlier [5], meaning detecting symptoms that the system's design or programming may be flawed [21]. As well, ML-based techniques are able to be used in design pattern recognition (adapter, strategy) [22]. Furthermore, some studies experimented with five design patterns (Singleton, Adapter, Composite, Decorator, and Factory Method) [23].

Some types of SDLC, for instance, Agile, divide the architecture of a system into components. Consequently, the selection of software components is part of the design phase. Some studies suggest a novel approach to machine learning [24], which can assist in the selection of reusable software components.

RQ2: The following machine learning models have been used for experiments in the selected studies: logistic regression, random forest, IBk [5], neural network and decision tree [22], zero, one, Naive Bayes, JRip, C4.5, SVMs (with different kernel functions), simple KMeans, and CLOPE [23].

The suggested machine learning approach to selecting reusable components combines the Decision Tree and Neural Network modules to determine the more accurate and suitable object of the software design pattern, which may help with efficient package reuse [24].

RQ3: The authors mentioned that the selected algorithms perform differently in terms of processing speed and classification accuracy [5], and they inform that Naive Bayes, Logistic regression, IB1, IBk, Random Forest have better performance than the VFI and J48 [5]. For instance, the authors in [22] inform us that the learning precision of the formulated approach is 67–95%

Generally, the ML-based techniques performed well in this phase. Nonetheless, the results are not compared with other traditional techniques (non-ML-based techniques). It's considerable to observe that although the researchers made an effort to provide impartial results, there may still be some degree of subjectivity, as long as all results are related to the construction of the training set, which is based on a manual design pattern labeling task.

4.3 Software construction

This phase involves turning the software design document into code using a programming language. It results in program code; thus, it is the logical one.

RQ1: The studies show that ML models are used for code generation [25, 26], documentation generation [27, 28], and code modification [29-31]. The popular models for converting ideas into code are ChatGPT, Codex, and Alphacode. ChatGPT and Codex are models by OpenAI. It interacts in a conversational way. As it is widely known, ChatGPT answers follow-up questions, challenges incorrect assumptions, and rejects improper demands. In addition, it is able to generate code [32]. Moreover, Codex is a general-purpose programming model, as it can be applied to any programming task [33].

Additionally, Alphacode is general-purpose programming, it can be applied to programming problems that require for deeper reasoning [34].

RQ2: The selected studies show that a wide range of ML techniques are able to be applied to various code generation tasks. The popular model types used by selected studies include recurrent neural networks [25-28] and convolutional neural networks [30]. ChatGPT is trained using supervision and reinforcement learning (RL). In the supervised learning, human trainers would provide conversations in which they played both sides, the user, and the chat bot side. Then, in the case of reinforcement learning, those people would be given the model-written responses to help them compose their response [32]. This dataset was combined with the Instruct GPT [35] dataset, which was converted to a question-answer format.

As well as Codex based on GPT-3, a neural network trained on text [35], this model has been trained on 179 gigabytes of Python code from software repositories hosted on GitHub projects. At the same time, Alphacode has been trained on 715.1 gigabytes of code on GitHub, in addition to Codeforces problems.

RQ3: In general, the outcomes were not assessed, considering more traditional techniques. Transformer models outperformed RNN models when the two were compared in an evaluative study [36]. Nonetheless, ML models perform imperfectly when evaluated on highly complex, unseen problems [32].

4.4 Software testing

Testing is defined as “an activity in which a system is executed under specified conditions, the results are observed or recorded, and an evaluation is made of some aspect of the system” (ISO/IEC 24765, 2006) [37].

In the software product development process, software testing is demanded. Any software product must first pass through several different steps before it can be implemented. Testing allows us to identify issues early. Additionally, participating in testing activities gives developers the ability to study the criteria for critical quality aspects, pose queries, and find solutions in advance.

Automation of software testing has been accepted as a realistic technique to get around the complexity and expense of most testing tasks. To find flaws in software systems, testing entails delving into their behavior. Applying machine learning (ML) to different software testing operations has drawn increasing interest [3].

RQ1: Machine learning was applied for statistical software testing [38], performance testing [39], and test case generation [40].

RQ2: Q-learning was used as a model-free RL algorithm in a smart test framework [39]. Furthermore, Model-Inference-Driven testing (MINTest) is used for software test automation [41]. It describes itself as a framework for unit and integration testing on its website [42].

RQ3: The studies show that efficient automated software testing is a challenging activity in software development [39-41]. The resulting test suites greatly improved in terms of defect detection [43].

4.5 Software maintenance

According to the IEEE Standard, IEEE STD 1219-15193 [44], software maintenance is: “the modification of a software product after its delivery (to the customer), to correct errors, to improve its performance or other attributes, or to adapt the product to a modified environment”.

Understanding software maintenance helps practitioners in the industry deal with many of the problems they currently experience, by reducing uncertainty, improving cost-effectiveness, dependability, and other factors [45]. This is the final stage of the SDLC. The software being produced is distributed to end users during this stage of the SDLC, who are then in charge of maintaining and utilizing it in accordance with best practices.

RQ1: The most dominant application of using ML in this phase is bug detection [46, 47].

In addition, maintenance software has several forms, for instance refactoring, which includes switching out components or algorithms for more elegant ones, updating data naming standards, and improving the readability or understandability of the code [47]. There are a few studies that discuss building a refactoring model, for instance, An AI-data-based approach to early quality evaluation and enhancement of object-oriented software products was proposed in the paper “A machine learning approach to software model refactoring” [48].

RQ2: Our study shows that a wide range of ML techniques have been applied in this phase. However, the CNN-based deep learning model is proposed for recognizing duplicate or similar bug reports [46]. Besides, three supervised machine learning algorithms are considered to build the model and predict the occurrence of the software bugs based on historical data by deploying the classifiers logistic regression, Nave Bayes, and decision tree [47].

A deep neural network that learns to detect the existence of functional decomposition in UML models of object-oriented software is used to implement model refactoring [48]. The study's proposed method [47] uses data science techniques to obtain an understanding of multidimensional software design aspects and then applies the knowledge acquired to generalize nuanced interactions between architectural elements [49-50].

RQ3: The selected studies don’t have very clear and effective evaluation methods. However, some studies show that some algorithms were able to generate 100% accuracy with train and test datasets [47]. On the other hand, the authors mentioned that the selected algorithm is empirically evaluated and shows high accuracy [48]. Furthermore, as with any ML model, the studies ensure that the results depend on the data. For instance, the proposed system in one of the studies provides a high accuracy rate for the same domain datasets and a low accuracy rate for different domain datasets [46]. In addition, the subjective nature of software affects the evaluation process [48].

5 Conclusion

Many authors have put significant effort into applying ML for SE. They were paying attention to give objective evaluations. However, it is able to contain some degree of subjectivity. Besides, it was quite significant to observe that this research study shows that a wide range of ML techniques can be applied to various phases of SDLC. One of the main conclusions is that applying machine learning algorithms correctly throughout the software development process is a very challenging task. Finally, we conclude that this study calls for the efficient cooperation venture between the ML and SE researchers to handle the open challenges.

References

1. L. E. Lwakatare, A. Raj, J. Bosch, H. H. Olsson, I. Crnkovic. *A taxonomy of software engineering challenges for machine learning systems: An empirical investigation*, in P. Kruchten, S. Fraser, F. Coallier (Eds.), Agile processes in software engineering and

extreme programming. XP 2019. Lecture notes in business information processing, Vol. 355, Springer, Cham, 227-243 (2019). doi: 10.1007/978-3-030-19034-7_14

2. M. Shehab, L. Abualigah, M. I. Jarrah, O. A. Alomari, M. S. Daoud MS. Artificial intelligence in software engineering and inverse. International Journal of Computer Integrated Manufacturing, **33(10-11)**, 1129-1144 (2020). doi: 10.1080/0951192X.2020.1780320
3. V. H. Durelli, R. S. Durelli, S. S. Borges, A. T. Endo, M. M. Eler, D. R. Dias, M. P. Guimaraes. Machine learning applied to software testing: A systematic mapping study. IEEE Transactions on Reliability, **68(3)**, 1189-1212 (2019). doi: 10.1109/TR.2019.2892517
4. F. Khomh, B. Adams, J. Cheng, M. Fokaefs, G. Antoniol. Software engineering for machine-learning applications: The road ahead. IEEE Software, **35(5)**, 81-84 (2018). doi: 10.1109/MS.2018.3571224
5. N. Maneerat, P. Muenchaisri. *Bad-smell prediction from software design model using machine learning techniques*, in 2011 Eighth international joint conference on computer science and software engineering (JCSSE), 11-13 May 2011, Nakhonpathom, Thailand, 331-336 (2011). doi: 10.1109/JCSSE.2011.5930143
6. P. Talele, R. Phalnikar. *Software requirements classification and prioritisation using machine learning*, in A. Joshi, M. Khosravy, N. Gupta (Eds.), Machine learning for predictive analysis. Lecture notes in networks and systems, Vol. 141, Springer, Singapore, 257-267 (2021). doi: 10.1007/978-981-15-7106-0_26
7. J. Zou, L. Xu, W. Guo, M. Yan, D. Yang, X. Zhang. *Which non-functional requirements do developers focuson? An empirical study on stack overflow using topic analysis*, in 2015 IEEE/ACM 12th working conference on mining software repositories, 16-17 May 2015, Florence, Italy, 446-449 (2015). doi: 10.1109/MSR.2015.60
8. A. Ahmad, K. Li, C. Feng, T. Sun. An empirical study on how iOS developers report quality Aspects on stack overflow. International Journal of Machine Learning and Computing, **8(5)**, 501-506 (2018).
9. C. Treude, O. Barzilay, M. A. Storey. *How do programmers ask and answer questions on the web? Nier track*, in 2011 33rd International conference on software engineering (ICSE), 21-28 May 2011, Waikiki, Honolulu, HI, USA, 804-807 (2011). doi: 10.1145/1985793.1985907
10. J. Zou, L. Xu, M. Yang, X. Zhang, D. Yang. Towards comprehending the non-functional requirements through developers' eyes: An exploration of stack overflow using topic analysis. Information and Software Technology, **84**, 19-32 (2017). doi: 10.1016/j.infsof.2016.12.003
11. A. Ahmad, C. Feng, K. Li, S. M. Asim, T. Sun. Toward empirically investigating non-functional requirements of iOS developers on stack overflow. IEEE Access, **7**, 61145-61169 (2019). doi: 10.1109/ACCESS.2019.2914429
12. H. Yin, D. Pfahl. *A preliminary study on the suitability of stack overflow for open innovation in requirements engineering*, in Proceedings of the 3rd international conference on communication and information processing, 24-26 November 2017, Tokyo, Japan, 45-49 (2017). doi: 10.1145/3162957.3162965
13. K. Bajaj, K. Pattabiraman, A. Mesbah. *Mining questions asked by web developers*, in Proceedings of the 11th working conference on mining software repositories, 31 May – 01 June 2014, Hyderabad, India, 112-121 (2014). doi: 10.1145/2597073.2597083

14. G. Pinto, F. Castor, Y. D. Liu. *Mining questions about software energy consumption*, in Proceedings of the 11th working conference on mining software repositories, 31 May – 01 June 2014, Hyderabad, India, 22-31 (2014). doi: 10.1145/2597073.2597110
15. M. Xiao, G. Yin, T. Wang, C. Yang, M. Chen. *Requirement acquisition from social Q&A sites*, in L. Liu, M. Aoyama (Eds.), Requirements engineering in the big data era. Communications in computer and information science, Vol. 558, Springer, Berlin, Heidelberg, 64-74 (2015). doi: 10.1007/978-3-662-48634-4_5
16. C. Rosen, E. Shihab. What are mobile developers asking about? A large-scale study using stack overflow. Empirical Software Engineering, **21(3)**, 1192-1223 (2016). doi: 10.1007/s10664-015-9379-3
17. Z. S. H. Abad, A. Shymka, S. Pant, A. Currie, G. Ruhe. *What are practitioners asking about requirements engineering? An exploratory analysis of social q&a sites*, in 2016 IEEE 24th international requirements engineering conference workshops (REW), 12-16 September 2016, Beijing, China, 334-343 (2016). doi: 10.1109/REW.2016.061
18. G. H. Pinto, F. Kamei. *What do programmers say about refactoring tools? An empirical investigation of stack overflow*, in Proceedings of the 2013 ACM workshop on refactoring tools, 27 October 2013, Indianapolis, Indiana, USA, 33-36 (2013). doi: 10.1145/2541348.2541357
19. A. G. Jivani. A comparative study of stemming algorithms. International Journal of Computer Applications in Technology, **2(6)**, 1930-1938 (2011).
20. A. Khan, B. Baharudin, L. H. Lee, K. Khan. A review of machine learning algorithms for text-documents classification. Journal of Advances in Information Technology, **1(1)**, 4-20 (2010). doi: 10.4304/jait.1.1.4-20
21. E. Fernandes, J. Oliveira, G. Vale, T. Paiva, E. Figueiredo. *A review-based comparative study of bad smell detection tools*, in Proceedings of the 20th international conference on evaluation and assessment in software engineering, 01-03 June 2016, Limerick, Ireland, 1-12 (2016). doi: 10.1145/2915970.2915984
22. R. Ferenc, A. Beszedes, L. Fulop, J. Lele. *Design pattern mining enhanced by machine learning*, in 21st IEEE international conference on software maintenance (ICSM'05), 26-29 September 2005, Budapest, Hungary, 295-304 (2005). doi: 10.1109/ICSM.2005.40
23. M. Zanoni, F. A. Fontana, F. Stella. On applying machine learning techniques for design pattern detection. Journal of Systems and Software, **103**, 102-117 (2015). doi: 10.1016/j.jss.2015.01.037
24. R. Selvarani, P. Mangayarkarasi. A dynamic optimization technique for redesigning OO software for reusability. ACM SIGSOFT Software Engineering Notes, **40(2)**, 1-6 (2015). doi: 10.1145/2735399.2735415
25. R. Agashe, S. Iyer, L. Zettlemoyer. Juice: A large scale distantly supervised dataset for open domain context-based code generation (2019). doi: 10.48550/arXiv.1910.02216
26. E. C. Shin, M. Allamanis, M. Brockschmidt, A. Polozov. *Program synthesis and semantic parsing with learned code idioms*, in 33rd Conference on neural information processing systems (NeurIPS 2019), Vancouver, Canada (2019).
27. A. Takahashi, H. Shiina, N. Kobayashi. *Automatic generation of program comments based on problem statements for computational thinking*, in 2019 8th International congress on advanced applied informatics (IIAI-AAI), 07-11 July 2019, Toyama, Japan, 629-634 (2019). doi: 10.1109/IIAI-AAI.2019.00132
28. Y. Shido, Y. Kobayashi, A. Yamamoto, A. Miyamoto, T. Matsumura. *Automatic source code summarization with extended tree-lstm*, in 2019 International joint conference on

neural networks (IJCNN), 14-19 July 2019, Budapest, Hungary, 1-8 (2019). doi: 10.1109/IJCNN.2019.8851751

29. M. Tufano, C. Watson, G. Bavota, M. D. Penta, M. White, D. Poshyvanyk. An empirical study on learning bug-fixing patches in the wild via neural machine translation. ACM Transactions on Software Engineering and Methodology, **28(4)**, 1-29 (2019). doi: 10.1145/3340544
30. Z. Zhu, Z. Xue, Z. Yuan. *Automatic graphics program generation using attention-based hierarchical decoder*, in C. Jawahar, H. Li, G. Mori, K. Schindler (Eds.), Computer vision – ACCV 2018. ACCV 2018. Lecture notes in computer science, Vol. 11366, Springer, Cham, 181-196 (2019). doi: 10.1007/978-3-030-20876-9_12
31. Y. Kim, H. Kim. *Translating CUDA to opencl for hardware generation using neural machine translation*, in 2019 IEEE/ACM international symposium on code generation and optimization (CGO), 16-20 February 2019, Washington, DC, USA, 285-286 (2019). doi: 10.1109/CGO.2019.8661172
32. R. Gozalo-Brizuela, E. C. Garrido-Merchan. ChatGPT is not all you need. A State of the Art Review of large Generative AI models (2023). doi: 10.48550/arXiv.2301.04655
33. M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. d. O. Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman, A. Ray, R. Puri, G. Krueger, M. Petrov, H. Khlaaf, G. Sastry, P. Mishkin, B. Chan, S. Gray, N. Ryder, M. Pavlov, A. Power, L. Kaiser, M. Bavarian, C. Winter, P. Tillet, F. P. Such, D. Cummings, M. Plappert, F. Chantzis, E. Barnes, A. Herbert-Voss, W. H. Guss, A. Nichol, A. Paino, N. Tezak, J. Tang, I. Babuschkin, S. Balaji, S. Jain, W. Saunders, C. Hesse, A. N. Carr, J. Leike, J. Achiam, V. Misra, E. Morikawa, A. Radford, M. Knight, M. Brundage, M. Murati, K. Mayer, P. Welinder, B. McGrew, D. Amodei, S. McCandlish, I. Sutskever, W. Zaremba. Evaluating large language models trained on code (2021). doi: 10.48550/arXiv.2107.03374
34. Y. Li, D. Choi, J. Chung, N. Kushman, J. Schrittwieser, R. Leblond, T. Eccles, J. Keeling, F. Gimeno, A. Dal Lago, T. Hubert, P. Choy*, C. de M. d'Autume, I. Babuschkin, X. Chen, P.-S. Huang, J. Welbl, S. Gowal, A. Cherepanov, J. Molloy, D. J. Mankowitz, E. S. Robson, P. Kohli, N. de Freitas, K. Kavukcuoglu, O. Vinyals. Competition-level code generation with alphacode. Science, **378(6624)**, 1092-1097 (2022).
35. B. Bhavya, J. Xiong, C. Zhai. Analogy generation by prompting large language models: A case study of instructGPT (2022). doi: 10.48550/arXiv.2210.04186
36. E. Dehaerne, B. Dey, S. Halder, S. De Gendt, W. Meert. Code generation using machine learning: A systematic review. IEEE Access, **10**, 82434-82455 (2022). doi: 10.1109/ACCESS.2022.3196347
37. H. Alaqail, S. Ahmed. Overview of software testing standard ISO/IEC/IEEE 29119. nternational Journal of Computer Science and Network Securit, **18(2)**, 112-116 (2018).
38. N. Baskiotis, M. Sebag, M. C. Gaudel, S. D. Gouraud. *A machine learning approach for statistical software testing*, in IJCAI 2007, Proceedings of the 20th International joint conference on artificial intelligence, 6-12 January 2007, Hyderabad, India, 2274-2279 (2007).
39. M. H. Moghadam, M. Saadatmand, M. Borg, M. Bohlin, B. Lisper. *Machine learning to guide performance testing: An autonomous test framework*, in 2019 IEEE international conference on software testing, verification and validation workshops (ICSTW), 22-23 April 2019, Xi'an, China, 164-167 (2019). doi: 10.1109/ICSTW.2019.00046
40. C. E. Tuncali, G. Fainekos, H. Ito, J. Kapinski. *Simulation-based adversarial test generation for autonomous vehicles with machine learning components*, in 2018 IEEE

intelligent vehicles symposium (IV), 26-30 June 2018, Changshu, China, 1555-1562 (2018). doi: 10.1109/IVS.2018.8500421

41. D. S. Battina. Artificial intelligence in software test automation: A systematic literature review. International Journal of Emerging Technologies and Innovative Research, **6(12)**, 1329-1332.
42. C. Rankin. The software testing automation framework. IBM Systems Journal, **41(1)**, 126-139 (2002). doi: 10.1147/sj.411.0126
43. L. C. Briand, Y. Labiche, Z. Bawar. *Using machine learning to refine black-box test specifications and test suites*, in 2008 The eighth international conference on quality software, 12-13 August 2008, Oxford, UK, 135-144 (2008). doi: 10.1109/QSIC.2008.5
44. *IEEE Standard for Software Maintenance*, in IEEE Std 1219-1993, The Institute of Electrical and Electronics Engineers, Inc., New York, 1-45 (1993). doi: 10.1109/IEEESTD.1993.115570
45. S. Levin, A. Yehudai. Towards software analytics: Modeling maintenance activities (2019). https://doi.org/10.48550/arXiv.1903.04909
46. A. Kukkar, R. Mohana, Y. Kumar, A. Nayyar, M. Bilal, K. S. Kwak. Duplicate bug report detection and classification system based on deep learning technique. IEEE Access, **8**, 200749-200763 (2020). doi: 10.1109/ACCESS.2020.3033045
47. S. D. Immaculate, M. F. Begam, M. Floramary. *Software bug prediction using supervised machine learning algorithms*, in 2019 International conference on data science and communication (IconDSC), 01-02 March 2019, Bangalore, India, 1-7 (2019). doi: 10.1109/IconDSC.2019.8816965
48. B. K. Sidhu, K. Singh, N. Sharma. A machine learning approach to software model refactoring. International Journal of Computers and Applications, **44(2)**, 166-177 (2022). doi: 10.1080/1206212X.2020.1711616
49. E. Akhmetshin, E. Klochko, I. Andryushchenko. A novel machine learning algorithms to assist traders and investors on forecasting stock market launches. Lecture Notes in Networks and Systems, **758,** 354-362.
50. I. S. Abdullaev, N. A. Prodanova, K. A. Bhaskar, E. L. Lydia, S. Kadry, J. Kim. Task offloading and resource allocation in iot based mobile edge computing using deep learning. Computers, Materials & Continua, **76(2),** 1463-1477(2023). doi: 10.32604/cmc.2023.038417