World Journal of Advanced Engineering Technology and Sciences

eISSN: 2582-8266

Cross Ref DOI: 10.30574/wjaets

Journal homepage: https://wjaets.com/

(Review Article)

Enhancing software engineering practices with generative AI: A framework for automated code synthesis and refactoring

Kodamasimham Krishna [*], Dheerender Thakur and Harika Sree Meka

Independent Researcher, USA.

World Journal of Advanced Engineering Technology and Sciences, 2024, 13(01), 672–681

Publication history: Received on 17 August 2024; revised on 26 September 2024; accepted on 29 September 2024

Article DOI: https://doi.org/10.30574/wjaets.2024.13.1.0463

Abstract

This paper is based on how software development has been revolutionized using AI in automation, mainly dealing with code synthesis and rewrite frameworks. While there is no focused definition for software development with AI technologies, their application in development processes is unlikely to remain marginal as they mature and provide higher productivity, improved code quality, and enhanced ability for automating repetitive tasks in development. Automating coding means that predefined tools can bring code suggestions, show and apply refactoring features, and enforce coding standards so developers can devote their efforts to other aspects of software development. The following paper describes the practical systems that should be adopted before AI can be incorporated into software development. It also covers essential factors such as ethical, security, and quality aspects regarding the proper use of artificial intelligence. Lastly, the paper focuses on the direction and innovations in the future, including domain-specific AI models, better explainability of the AI solutions, and collaborative tools that can work according to improved and modified development practices. The framework is intended to balance machine learning with human experience to help developers utilize the benefits that an AI-Based Software Development Framework can render.

Keywords: AI-driven automation; Code quality; Ethical considerations; Machine learning; Software development

1. Introduction

Software engineering has always been an area where art meets 'science.' Application developers are expected to conceive, implement, and fine-tune application code marked by functionality and performance characteristics. With the increase of the current scale and the enhancement of the software systems, the pressures on the developers have also increased. The problem is that even though older and more classical practices might be effective, they still need to work entirely in today's fast-paced software development. Activities include synthesis, where developers write code from the ground up in light of particular specifications, and refactoring, a process of improving existing code or restructuring. The Process is usually exhaustive and tends to be marred by errors inherent in any human activity. These tasks provide enough quality for the developed software; however, they become bottlenecks of the development process and result in negative consequences such as inefficiency and technical debt. This raises an important question: how can we fix and improve the rates of such fundamental software engineering activities without affecting their accuracy?

In recent years, the answer has gradually surfaced as generative artificial intelligence (AI). By applying Generative AI along with NLP and Machine Learning capabilities, it has been witnessed that AI can understand simple function signatory or even high-level intent to write a program and generate code accordingly. Some pristine examples include GitHub's Copilot and OpenAI's Codex, which has already stirred things up in the software engineering circles by suggesting the piece of code that comes next and even autocompletion of code depending on the context. These intelligent tools enable coders to practice more creativity and productivity within a shorter time frame than was once

[*] Corresponding author: Kodamasimham Krishna

possible- but also enable developers to concentrate on the more advanced and essential levels of coding while allowing automation to handle the regular and tedious tasks. Nevertheless, the opportunities for generative AI are evident; there are both critical issues and some gaps in modern generative AI when it comes to code synthesis and refactoring.

Currently, somewhat limited attention is paid to the use of generative AI in the software engineering lifecycle such that code generation and refactoring are performed without compromising the quality and extensibility of the software. Despite this, current AI tools allow for simple, narrow tasks to be executed effectively, including writing code; however, when it comes to more complex software development issues or even those that require a deeper contextual understanding and taking into consideration long-term code maintainability, AI's stay behind. For instance, AI may be good at generating the boilerplate of code or presenting suggestions for small codes, but to make sure that the code it generates blends well into larger multi-module systems is more accessible said than done. Also, they do not perform well in technical debt handling, especially when working with large and dynamic code reservoirs that require rewiring for future gains. The current approaches to AI solutions need to allow for the identification and addressing of some of the issues at the architectural level, namely underlying system designs and algorithms that may severely affect the scalability and performance of program systems when implemented.

About the successive points that have contributed to the definition of software engineering, a second gap is that of the human supervision still necessary for using AI in development. Despite all the advantages for developers nowadays implementing AI code generation, the code generated by AI needs to be reviewed, edited, and integrated into the current system. Additionally, it is also pointed out that AI does help with refactoring. Still, it does not have a general knowledge of the overall system as the developers do. Such knowledge is often necessary to make sensible decisions on how to prepare code for the system's future growth. The absence of sophisticated refactoring technologies that endow current generative AI options with the capacity to make these decisions demonstrates the need for current generative AI models in solving these annual software development issues.

In addition, incorporating AI brings about new issues with security and reliability in development lifecycles. Code created by the AI while being functional can have vigilance issues leading to the creation of security vulnerabilities, or it may not follow some of the best practices. This is because artificial intelligence must be closely supervised in large systems where even minor vulnerabilities may generate a series of consequences. That is why the problem is still relevant, and the industry has not yet developed broad solutions that integrate AI into software development and ensure the generated and refactored code is secure, scalable, and conforms to ideal coding standards.

Based on these issues, this paper recommends a framework that improves software engineering methodologies by integrating generative AI technology for auto-generating codes and intelligent code transformation. As the present paper suggests, the proposed framework enhances AI's capacity to comprehend the software's architecture. Improving code quality and making it more maintainable with less human interference is possible. Using the ideas of modern advanced models, feedback, and integration into the development environments, this idea can bring several benefits to the new software engineering approach to overcome the difficulties of modern software systems.

2. Background and context

Therefore, It is imperative to look at the conventional methodologies, which have been the bedrock of software development, to fully grasp the impact that interaction with an automated AI system holds for the development discipline. Conventional software development involves several vital stages: requirements analysis, design, implementation, testing, and maintenance. Each stage is time-consuming and involves close cooperation between the team members to meet end-users' requirements and ensure the software works properly. Though these processes are highly formalized, sometimes intent human error, repetitive coding work, and problems with code quality pervasiveness emerge. Manual code synthesis refers to a process in which developers type codes as per requirement specifications one line at a time, and such a process is often slow and error-prone. Like in the case of automatic refactoring, restructuring the source code to enhance its readability, speed of execution, and ease of maintenance is complex and demands profound knowledge in the programming and usage of the code.

Over the past few years, integrated AI technologies have emerged in software development to solve these challenges. Language processing, machine learning, and deep learning are among the AI technologies extensively discussed in SDLC process automation. For example, machine learning can learn from big code data to establish a pattern and write new code. At the same time, NLP can be employed to read and understand natural language descriptions of required functionality and translate the same into code. Such technologies have culminated into various tools and frameworks that enable developers to enhance and simplify coding practices using artificial intelligence.

It is possible to distinguish several significant turning points in the development of AI as a branch of software engineering. The first approaches were made in 1991 with attempts to develop rule-based systems that assist in simple coding. However, these systems could have been more sophisticated and could only offer essential support. The actual advancement came with the emergence of machine learning and deep learning techniques, which enabled systems to learn from vast quantities of data and draw well-reasoned conclusions. Recent products that use deep learning models to predict code completions and then type out snippets of code, namely the GitHub Copilot and TabNine, are rapidly becoming very popular among developers. These tools shorten the coding process and prevent mistakes, but they also assist the developer by offering suggestions based on context compliant with the industry standards.

Still, specific problems arise from the attempts to launch AI into the software development process. Another is to prevent the code developed using AI from being less reliable than it should be for professional software development. Even though the models are highly advanced, human employees should still be wary of them since they can generate syntax-correct but logically incorrect code or even inject security flaws. Additionally, there is the challenge of domain specificity: When the models have been trained on general data sets, they are likely to fail when it comes to specific domains of software development where there is a need to understand specific technologies or ways of implementation that are peculiar to the particular area in consideration.

Furthermore, the main issue of incorporating AI-based tools into conventional software development processes is the strategies employed. Developers must rely on these tools to implement their implementation, which means that AI tools must explain why they recommend specific steps. This is important in creating trust among the developers since they need to know how the suggestions from the AI system are arrived at so that they can, in turn, determine the suitability of those suggestions for their task.

In the same way, with the development of artificial intelligence in the future, the usage of AI in software development will be maximized, and more exclusive and enhanced methods will be provided to software developers. Yet, to unlock the full potential of AI-based automation, it is critical to establish effective best practices that will frame the integration of the former into an SDLC and, in effect, strengthen the synergy with human expertise rather than legitimize it. Only if the traditional problems of software development and the possibilities of AI technologies are paid attention to will the idea of automation by applying AI be appreciated and the steps to implement it be found.

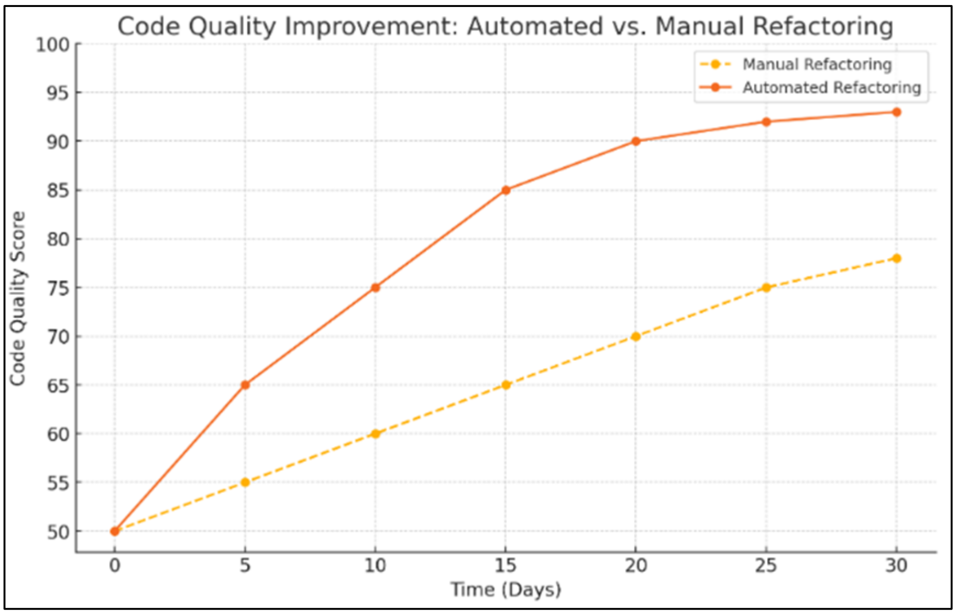

Figure 1 Comparison between automated and manual refactoring in terms of code quality improvement over time

3. AI-driven code synthesis

Code synthesis, the process of writing code with the help of specified requirements or specifications, is one of the primary constituents of software developments to which AI contributes more and more. The idea of code synthesis was that developers wrote the code by themselves with the help of their interpretation of the specifications. The process consumes a lot of time and is ridden with mistakes. Code synthesis with the help of AI is another approach that tries to separate writing descriptions of the code and make actual code automatically using machine learning algorithms capable of understanding given definitions and finding templates that should be used. The current methodology is anticipated to enhance productivity further and alleviate the cognitive burden associated with incentives that concentrate developers on the process's more creative and challenging aspects, specifically software design.

Automatically generated code employs different machine learning methods to generate code. One of their most powerful is based on deep learning models, such as transformers and recurrent neural networks (RNNs), which are trained by the large code corpus that teaches the model's syntax, semantics, and coding standards. These models can train a prediction of the following line of code or complete functions given a partially completed code or a description in natural language. For example, a developer may describe a function they want to be implemented, and the AI system writes the code, considering the programming language, libraries, and frameworks being used.

Besides deep learning, reinforcement learning has become one of the most influential techniques for code synthesis. In the case of reinforcement learning, the models are trained for the code generation process, where they get a reward for the correct and quality code. This approach helps the model make a better output than before because it adjusts itself with the help of server logs and the mistakes that occur in the model. With the help of deep learning in parallel with reinforcement learning, AI systems can generate syntactically and semantically correct code and optimize its performance, legibility, and maintainability.

This AI code synthesis uses a range from one sector to the other. One of the most tangible is that of particular applications helpful in automating the generation of what can be called 'canned' code. Accepted code, whereby the latter represents a set of ideal source code, repetitive sections, usually with slight variation between repetitions, could be more exciting and time-consuming for developers to write and maintain. AI can generate This code automatically, thus saving time on the need to write code, which is quite repetitive for developers. Moreover, with the help of AI code generators, creating several prototypes becomes possible, which means faster development of the best option.

As discussed below, AI-driven code synthesis has advantages, disadvantages, and limitations. One is the ability to generate quality code free from security vulnerabilities. Even though the AI models can produce syntactically correct code, they are several times far from producing best practice-compliant code or containing no security flaws. This issue occurs mainly because designing AI models requires more context and knowledge of the task and environment than the developers do. For instance, the model trained on a software code for average software may not write quality code for complex software such as those used in embedded systems and high-frequency trading platforms due to complexities inherent in the latter.

Another challenge is in the appropriateness of deploying the AI-driven code synthesis tools and the interpretability of the synthesized code. To become a relevant feature in developers' toolchains, they need to be able to rely on them. However, the tendency of many AI models to be 'black boxes' makes it difficult for people to understand how the code is being generated and why those specific suggestions can significantly decrease trust and, subsequently, the tool's usage. To this end, research is being conducted to create systems that generate machine code together with other supporting information on how the code should be written and why the output for the code was generated and to enable the programmer to decide whether to accept or reject the generated code.

However, incorporating AI's code synthesis in the current development interfaces must be done carefully in the correct implementation plans. Based on this, technology tools must not be an addition that disturbs work processes but rather an improvement that can integrate with common IDEs and version control systems. Further, some costs are related to institutionalizing new technologies and AI-driven tools every organization must implement. These require resources to overcome this hurdle and provide adequate training to the developers.

Therefore, code synthesis based on artificial intelligence is a significant innovation in the sphere of programming that contributes to the automation of the work of programmers, boosting their performance and decreasing the possibility of errors. These tools include AI writing software that employs deep learning, reinforcement learning, and numerous other forms of artificial intelligence to assist in translating natural language into code that conforms to best practices and different stipulated standards. However, to attain those benefits in the application of AI in code synthesis, it is imperative to consider the issues related to quality assurance, security, the disclosure of the synthesis process, and their integration. Algorithms and related technologies are bound to become more pervasive in the software development lifecycle; the developers will be able to concentrate on unconventional thinking.

4. AI-driven refactoring

Refactoring is defined as the act of restructuring already written code in such a way that enhances its comprehensiveness, efficiency, and flexibility. At t, the same time, the called external procedure stays the same. It is an essential process in computer programming and software engineering, as it enables the teams to control technical debt and ensure that code repositories are clean and optimized as much as possible. Conventionally, refactoring is wrought with manual operations, which consumes a lot of time, and the intervention of an expert professional is needed to determine sections that can undergo improvements and consider the safety margins of the operations. However, refactoring is currently emphasized as one of the critical processes that can benefit from the advanced solution of AI refactoring. It is possible to detect code smells and recommend and perform optimizations and refactoring using supervised and unsupervised machine learning and data analysis techniques while preserving the functional correctness of the source code.

AI-based refactoring is done by scanning the structures and trying to comprehend which may be appropriate for refactoring. Attending to generality, different heuristics can be learned by the models, such as detecting duplicated code, large classes, long methods, and other symptoms of innate poor software quality, known as 'code smells.' Learnability: such models can also incorporate knowledge, such as language-specific idioms, framework recommendations, and overall software architectures. It also allows AI to propose specific refactoring actions most relevant to current best practices.

It is convenient to perform automated code transformations, making the AI refactoring tool one of the most powerful. After potential improvements are found, AI can apply the refactoring to the code if left alone. For instance, an AI tool may automatically refactor source code by cutting an extensive method into more manageable functions and providing descriptive names to variables. Alternatively, the AI may simplify the arrangement of conditionals. They are integrated at the source code level as this is the most efficient way, which takes much less effort from developers and does not consume as much time as when a human is involved. Also, using AI-based refactoring tools, there will always be more places to find for refactoring new or updated software for better optimization as development goes on.

The proposed refactoring process underlined above has several advantages when based on AI. First, it allows for keeping the code quality high, constantly looking for imperfections that could be potential problems. This approach minimizes technical debt, which could accrue with time as codebases expand and may slow development, as much effort is spent on fixing previous technical debts. AI tools also help automate the tedious work of refactoring tasks, where developers can get bored doing repetitive code maintenance work all the time and move ahead with more creative and crucial jobs related to software development. Also, it is essential to note that the optimization of the code paths shall be done automatically from the refactoring tools, which can benefit the software by reducing its resource utilization and response time, hence improving the users' experience.

However, it is worth noticing that implementing the idea of refactoring based on artificial intelligence also poses several potential problems. The first concern is that automated refactoring should avoid introducing faults and other side effects. Each AI model can offer intelligent suggestions about specific patterns and generally accepted conventions. Still, it will not have the reasons for a particular application or a broader impact of some modifications. For instance, optimizing code by refactoring, such as improving a code's readability, could impair performance or alter the behavior of related components. Proper testing and validation must often be carried out to prevent potentially negative impacts of refactoring on a software’s functionality and reliability from occurring.

Another issue is the ability to fit AI-driven refactoring tools into existing development processes as the last point. These tools must be easily incorporated into well-used IDEs and version control software to be maximally useful. It must offer interactive panels where the developers can scrutinize and endorse the refactoring suggestions efficiently to guarantee that all the alterations are in harmony with the team's coding specifications and the project's needs. Moreover, one should see how the particular AI models make their refactoring choices, and it has to become reliable as developers rely on those tools. Expl AI techniques can assist by giving the designer an explanation as to why a particular refactoring action is suggested, as well as how the action should be implemented.

AI-driven refactoring represents a significant advancement in the software development lifecycle, offering the potential to automate routine maintenance tasks, improve code quality, and reduce technical debt. By leveraging machine learning algorithms and data-driven techniques, these tools can detect code smells, suggest optimizations, and perform automated refactoring, all while preserving the integrity of the software. However, to ultimately capitalize on the advantages of AI-driven refactoring, it is essential to address the obstacles of ensuring functional correctness, managing integration into existing workflows, and fostering trust among developers.AI technologies will probably become more significant in the maintenance and enhancement of codebases, thereby allowing teams to deliver high-quality software more efficiently and effectively.

5. A framework for AI-driven code synthesis and refactoring

Having integrated contemplation of the AI applications in code synthesis and code refactoring is crucial to approaching these technologies' potential. When the market is filled with innovative and intelligent tools, such a framework helps to think within a modern corporate framework, creating a structure for implementing AI techniques together with traditional practices, preventing deterioration of code quality, and striving for permanent improvement of efficiency based on intelligent solutions. The associated framework should be constructed to cater to development environments of different complexities and, simultaneously, can be easily updated by new AI technologies and practices.

The first part of this framework is data collection and data cleaning. Regarding synthesis and refactoring of code, it is vital to recognize that improvements in AI-based techniques are primarily determined by the quality and variety of the data used to train correlating models. This stage includes collecting a vast and diverse set of source code, which covers diverse programming languages and different stylistic approaches and belongs to other domains. This way, the data has to be selected to contain not just quality code that is easy to read and adherent to best practices but also examples of the anti-patterns and standard errors. The data should also be categorized as much detail as possible to identify concrete coding practices, architectural patterns, and refactoring. Cleaning involves removing irrelevant data from the data set, normalizing, and transforming the data into the desired format and scale. At the same time, annotating makes the data easily understandable by the machine learning model. This step is crucial because the quality of the input data guides the process of AI models for code synthesis and refactoring.

The second consists of model training and admissions decision evaluation. Once the data is prepared, the next step is to build machine learning models to execute predetermined code synthesis and refactoring tasks. These models can be funded with different AI techniques like deep learning techniques, reinforcement learning techniques, or a hybrid model. But, when the models are trained, they can find probable patterns in the data, estimate the following line of code, suggest an improved version, and detect likely refactoring opportunities. It is essential in any model development where model evaluation is done periodically to check if the models developed are good and can generalize well with other unseen data. Here, the test datasets are utilized to test the models, and factors like accuracy, precision, recall, and F1-score are used to determine the models' efficiency. It should also be mentioned that to enhance the algorithm's performance, it is possible to implement quantitative and qualitative evaluations, including code reviews by professionals, to evaluate the applicability and quality of the generated code and refactoring advice.

Deployment and integration make up the final concept of the framework. AI models must be deployed in real development contexts when the training and evaluation processes are completed. This step includes incorporating the AI tools in the commonly used IDEs and other related tools, such as the version control systems and CI/CD processes. Thus, the objective is to provide developers with AI-generated recommendations and automate code refactoring as integrated services within developers' current workflows. To this end, the tools must offer easily usable interfaces and provide or suggest simple developer recommendations. Further, the tools should incorporate configurable options to enable teams to set the extent to which the tools are automated, as well as the type of suggestions required.

This means that the process certainly has a scientific basis, and the last element of the framework is the feedback loop and continuous improvement. Tools that utilize AI to perform code synthesis and refactoring must not be fixed; they have to be updated over time upon receipt of feedback from the developers and changes in coding conventions. This, in turn, demands the existence of a sound feedback mechanism through which the developers can give their feedback on the suggestions given by the AI tools and the automated modifications made to it. Users can provide direct input by reviewing codes and evaluating the changes to the code and their impact on the software's performance and simplicity of maintenance. This information can then be used to retrain the models and make the subsequent models even finer to serve the purpose and intention for which the models are designed on a long-term basis. Furthermore, it provides a way to find out any problems or constraints in the AI models to enable the developers to fix them and enhance the quality and accuracy of the tools.

The suggestions, when used to support this or a similar kind of framework, must be well planned out and executed. It signifies that organizations must focus on choosing the proper AI techniques and models depending on their requirements and codebases. Another factor that should be considered is the need for training and support so that the developers are at ease when it comes to the new tools and are in a position to integrate the new AI into their development cycles. In addition, organizations should provide guidelines and templates that can be adopted when one wants to use these tools since the use of these tools should be balanced with a human touch when evaluating people.

6. Ethical, security, and quality considerations

Incorporation of AI automation in software development is an interesting idea that Provides several ethical, security, and quality concerns that should be considered when applying this technology. On the one hand, the implementation of AI tools greatly benefits production in terms of improvement of efficiency and productivity; on the other hand, it raises potential risks and challenges which might affect the quality of the targets and the purity of the processes within software development. There is, however, a need to address these concerns if authors are to derive the total impact of synthetically synthesizing all forms of codes Ref. , [15].

Some of the most relevant ethical issues that relate to the use of AI-based tools in software development include the following: The first is bias and opacity. Training an AI model with biased data will result in bias exhibited at the output level, thus becoming unfair or discriminating in software development. For instance, if the training data only provide examples of coding styles or practices of a particular culture or gender, the AI will tend to favor such practices to the detriment of the others, thus having potential bias effects. To address this risk, though, the data used for training AI models must span across many types of code and styles and various domains and practices. Further, AI models should be mimicked in nature and must be able to explain why they recommend a particular change. This is important and useful to the developers in getting the rationale behind the implemented AI functions and ways in which the code refactoring is being suggested so that appropriate action can be taken.

Security is another possible problem when employing AI tools in software development applications. Suppose the given model can write code or write code in such a way that it may even automatically refactor the code. In that case, the given model might be introducing new vulnerabilities if the model does not understand what it is doing. For instance, an intelligent tool may offer changes in the code that create security vulnerabilities, poor management, or improper input information storage. Regarding such issues, it is crucial to ensure that security testing and validation issues are incorporated into the application of AI in the development cycles. This is useful in preventative measures where possible risks are detected using static and dynamic analysis tools; there is a security audit of all the code that has been created by the AI from time to time. It is useful to review all the developers' changes before committing them to the base.

Quality is another critical issue when adopting the code synthesis and refactoring technology that AI powers. While using these tools is very useful in increasing the efficiency of work and delegating monotonous tasks to the tools, it must be noted that these tools are only sometimes fully efficient or may generate the code that needs to be properly formatted as per professional coding standards. It means that using AI to write code might result in syntactically correct code, which in no way can offer the standards that an experienced programmer will consider optimum in terms of code readability, maintainability, and optimizations. Some of the ways to extend the proposed approaches and raise the quality of code generated by machine learning tools are testing and code review. These include the unit tests used to test what the coding produces, integration tests used to test the blocks of codes joined together, and lastly, the pe, the performance t, tests used to test the rate at which the code works. In addition, getting feedback from developers from time to time also helps improve the AI models to provide better quality.

Other questions should be mentioned and solved when using AI in software development, such as legal and regulatory ones. This involves consideration of the country's rules in the inscription of personal data, such as the GDPR laws of the European Union. The code base used in developing the AI models contains personal or sensitive information. It should be used with a lot of precautions in order not to offend privacy policies. More specifically, there should be clear regulations and processes of how AI should be employed to achieve the above legal purposes so that all the created tools employ AI responsively and within the legal confines.

Therefore, it is possible to conclude that new paradigms for CS, such as code refactoring and AI-based ones, are effective in SW development. Still, they must consider the ethical, security, and quality issues caused by their application. That means creating representative training data and investing in enough security and quality control mechanisms and guidelines; the AI must explain and be transparent, supporting the ethical frameworks of AI and people's continual engagement to learn and work together: organizations are capable of doing the right thing and getting the most from the possibilities of AI. These considerations will become important in the future of SW developments since the mentioned tendencies correlate with future AI automation's impact on productivity and creativity and on creating the most ethical, safe, and high-quality world.

7. Future directions and innovations

Several innovations and trends expected to take shape soon will define the future of software development automation using artificial intelligence. Therefore, the mandate of AI technologies to continue growing will bring new ways of increasing the functionality of AI-powered tools, thereby making them more robust, versatile, and universal to the SDLC.

One potential area of expansion is the developmental progression of these models to incorporate mechanisms of greater sophistication as applied to the actualization of software. Modern AI-based code synthesis and refactoring tools are very efficient in handling routine tasks in the analysis and organization of code in terms of patterns for reusing, etc. However, a lot more can be done. Subsequent AI implementations may be achieved with stronger cognition of the structural layout of dependable software systems to make more informed architectural decisions concerning software elements, dependencies, and interactions. This would allow some of the more intricate refactoring activities to be aided by the employment of AI on different levels, for example, dealing with monolithic applications, converting them to microservices, or optimizing the code for parallelism in distributed systems.

One more emerging area for the application of AI is the combination of AI with expertise in a specific field and industry. As general-purpose repositories predetermine the current models, future developments could concentrate on bringing in approaches with prior knowledge in particular fields, like finance, medicine, and aerospace. Despite this, integrating domain-specific conventions and standards into AI tools could enhance the tools' performance and relevance in helping to generate and optimize code with regard to various industry-specific parameters. Such an approach would extend the role of applying AI tools to more specific software development environments, in which the context of an application is a vital factor to consider while developing the code.

Together with the domain-specific knowledge, the factors that are expected to underlie future AI-based instruments include explainability and transparency. One of the current issues about AI models, especially the deep learning models, is that the models make suggestions and changes. Still, it takes a lot of work for developers to know why a specific suggestion has been made. Technological advancements in explainable AI (XAI) attempt to move in this direction by proposing ways and means to furnish humans with understandable reasons for AI-based output. By increasing transparency in the functioning of AI, developers learn to trust AI tools and work in conjunction with them much more often, thus providing robust and accepted results.

Moreover, it is expected that directions in the development of AI automation will push the boundaries of continuous learning and optimization. Present-day AI models usually need to be retrained on some schedules to update new data and reflect the alterations in coding norms. Future models could use online learning, meaning that the models could learn from newer code changes, developer feedback, and, generally, newer software development practices. This capability would help AI-driven tools be aware of the trends in development and offer the current state-of-the-art suggestions for the synthesis of code as well as code restructuring.

8. Conclusion

The emergence of artificial intelligence in software development is paving the way for new approaches capable of improving software development in terms of speed, quality, and productivity using new code synthesis and transformation methods. These tools incorporate machine learning models and data science techniques to do mundane work, detect the presence of smell indicators, and improve the code structure so that developers can work on more challenging and innovative aspects of their projects. AI technology will progress further and become an active part of software development practices as it can write more accurate and contextually relevant codes and refactor the existing codes.

However, there are several essential issues to consider when implementing artificial intelligence for automation at total capacity. Ethical considerations, including bias and explainability, need to be addressed to have proper ethical AI outcomes. Potential risks include introducing security threats by implementing this system, and security considerations, in this case, have to be handled with care through proper validation and testing of the system. Further, the constant focus is placed on maintaining basic code quality standards to ensure that both AI-written and refactored code is acceptable and safe in the eyes of developers and end-users.

As such, the future of AI in software development shall continue to be more innovative and progressing, creating better AI algorithms that work with the various substantial structures involved in software, including the incorporation of domain knowledge, better explainable AI, and better teamwork and collaboration tools, especially for distributed teams.

Updating and usefulness will also help maintain a continuous understanding of new coding approaches and the new technologies in the market.

Compliance with ethical standards

Disclosure of conflict of interest

No conflict of interest to be disclosed.

References

[1] Rahman, M. A. (2024). Optimization of Design Parameters for Improved Buoy Reliability in Wave Energy Converter Systems. Journal of Engineering Research and Reports, 26(7), 334-346.

[2] Rahman, M. A., Uddin, M. M., & Kabir, L. (2024). Experimental Investigation of Void Coalescence in XTral-728 Plate Containing Three-Void Cluster. European Journal of Allamanis, M., Barr, E. T., Devanbu, P., & Sutton, C. (2018). A survey of machine learning for big code and naturalness. ACM Computing Surveys (CSUR), 51(4), 1-37.

[3] Chen, T., Liu, Z., Li, X., & Shi, Y. (2019). Automated code refactoring for ensuring security in blockchain smart contracts. Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE '19), 53-64.

[4] Fregnan, E., Zeni, M., Scalabrino, S., Beller, M., & Bacchelli, A. (2020). Automated code review: A systematic literature review. IEEE Transactions on Software Engineering, 48(4), 1-22.

[5] Lata, R., Akshatha, A. R., & Garg, S. (2020). The impact of artificial intelligence on software development processes. Journal of Systems and Software, 169, 110710.

[6] Li, Y., Guo, L., Jin, Z., & Wang, X. (2021). AI-based code completion: A review of challenges and solutions. IEEE Access, 9, 143299-143313.

[7] Ray, B., Hellendoorn, V., Godhane, S., Tu, Z., Bacchelli, A., & Devanbu, P. (2016). On the "naturalness" of buggy code. Proceedings of the 38th International Conference on Software Engineering (ICSE '16), 428-439.

[8] Tufano, M., Watson, C., Bavota, G., Pantiuchina, J., Poshyvanyk, D., & Oliveto, R. (2019). An empirical investigation into the nature of automated code changes. Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering (ICSE '19), 1-12.

[9] White, M., Vendome, C., Linares-Vásquez, M., & Poshyvanyk, D. (2016). Toward deep learning software repositories. Proceedings of the 12th Working Conference on Mining Software Repositories (MSR '15), 334-345.

[10] Xu, F., Chen, L., & Liang, X. (2021). A machine learning-based approach for improving software quality by automated refactoring. Journal of Software: Evolution and Process, 33(6), e2357.

[11] Zhai, S., Gu, T., Ke, Q., Xu, X., & Sun, H. (2020). Towards improving software maintainability with AI-driven automated refactoring. IEEE Transactions on Software Engineering. Advanced online publication.

[12] STEM fields. International Journal of All Research Education and Scientific Methods, 11(08), 2090-2100.

[13] Engineering and Technology Research, 9(1), 60-65.

[14] Rahman, M. A. (2024). Enhancing Reliability in Shell and Tube Heat Exchangers: Establishing Plugging Criteria for Tube Wall Loss and Estimating Remaining Useful Life. Journal of Failure Analysis and Prevention, 1-13.

[15] Murali, S. L. ADVANCED RRAM AND FUTURE OF MEMORY.

[16] Chen, Y., & Li, C. (2017, November). Gm-net: Learning features with more efficiency. In 2017 4th IAPR Asian Conference on Pattern Recognition (ACPR) (pp. 382-387). IEEE.

[17] Elemam, S. M., & Saide, A. (2023). A Critical Perspective on Education Across Cultural Differences. Research in Education and Rehabilitation, 6(2), 166-174.

[18] Rout, L., Chen, Y., Kumar, A., Caramanis, C., Shakkottai, S., & Chu, W. S. (2024). Beyond first-order tweedie: Solving inverse problems using latent diffusion. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (pp. 9472-9481).

[19] Thakur, D. & IRE Journals. (2020). Optimizing Query Performance in Distributed Databases Using Machine Learning Techniques: A Comprehensive Analysis and Implementation. In IRE Journals (Vol. 3, Issue 12, pp. 266–267) [Journal-article]. https://www.irejournals.com/formatedpaper/1702344.pdf

[20] Krishna, K. (2020). Towards Autonomous AI: Unifying Reinforcement Learning, Generative Models, and Explainable AI for Next-Generation Systems. Journal of Emerging Technologies and Innovative Research, 7(4), 60–61. https://www.jetir.org/papers/JETIR2004643.pdf

[21] Murthy, N. P. (2020). Optimizing cloud resource allocation using advanced AI techniques: A comparative study of reinforcement learning and genetic algorithms in multi-cloud environments. World Journal of Advanced Research and Reviews, 7(2), 359–369. https://doi.org/10.30574/wjarr.2020.07.2.0261

[22] Murthy, P. & Independent Researcher. (2021). AI-Powered Predictive Scaling in Cloud Computing: Enhancing Efficiency through Real-Time Workload Forecasting. In IRE Journals (Vol. 5, Issue 4, pp. 143–144) [Journal-article]. https://www.irejournals.com/formatedpaper/1702943.pdf

[23] Mehra, N. A. (2021). Uncertainty quantification in deep neural networks: Techniques and applications in autonomous decision-making systems. World Journal of Advanced Research and Reviews, 11(3), 482–490. https://doi.org/10.30574/wjarr.2021.11.3.0421

[24] Mehra, A. D. (2020). UNIFYING ADVERSARIAL ROBUSTNESS AND INTERPRETABILITY IN DEEP NEURAL NETWORKS: A COMPREHENSIVE FRAMEWORK FOR EXPLAINABLE AND SECURE MACHINE LEARNING MODELS. International Research Journal of Modernization in Engineering Technology and Science, 2.

[25] Krishna, K. (2022). Optimizing Query Performance In Distributed Nosql Databases Through Adaptive Indexing And Data Portioning Techniques. In International Journal of Creative Research Thoughts (IJCRT), International Journal of Creative Research Thoughts (IJCRT) (Vol. 10, Issue 8) [Journal-article]. https://ijcrt.org/papers/IJCRT2208596.pdf

[26] Krishna, K., & Thakur, D. (2021). Automated Machine Learning (AutoML) for Real-Time Data Streams: Challenges and Innovations in Online Learning Algorithms. In Journal of Emerging Technologies and Innovative Research (JETIR), Journal of Emerging Technologies and Innovative Research (JETIR) (Vol. 8, Issue 12) [Journal-article]. http://www.jetir.org/papers/JETIR2112595.pdf

[27] Murthy, P., Thakur, D., & Independent Researcher. (2022). Cross-Layer Optimization Techniques for Enhancing Consistency and Performance in Distributed NoSQL Database. International Journal of Enhanced Research in Management & Computer Applications, 35. https://erpublications.com/uploaded_files/download/pranav-murthy-dheerender-thakur_fISZy.pdf

[28] Murthy, P., & Mehra, A. (2021). Exploring Neuromorphic Computing for Ultra-Low Latency Transaction Processing in Edge Database Architectures. Journal of Emerging Technologies and Innovative Research, 8(1), 25–26. https://www.jetir.org/papers/JETIR2101347.pdf

[29] Mehra, A. (2024). HYBRID AI MODELS: INTEGRATING SYMBOLIC REASONING WITH DEEP LEARNING FOR COMPLEX DECISION-MAKING. In Journal of Emerging Technologies and Innovative Research (JETIR), Journal of Emerging Technologies and Innovative Research (JETIR) (Vol. 11, Issue 8, pp. f693–f695) [Journal-article]. https://www.jetir.org/papers/JETIR2408685.pdf

[30] Thakur, D. & IJARESM Publication. (2021). Federated Learning and Privacy-Preserving AI: Challenges and Solutions in Distributed Machine Learning [Journal-article]. International Journal of All Research Education and Scientific Methods (IJARESM), 9(6), 3763–3764. https://www.ijaresm.com/uploaded_files/document_file/Dheerender_Thakurx03n.pdf