Published online: 19 November 2024 DOI: 10.54254/2977-3903/13/2024141

Current applications and future prospects of artificial intelligence in software engineering

Zherong Ma

Guangzhou ULink International School, Guangzhou, 511466, China

3230817511@qq.com

Abstract. The advent of digital technology has spawned a revolution in software engineering, with artificial intelligence (AI) emerging as a key technology. The integration of advanced techniques, such as natural language processing (NLP) and deep learning has demonstrated AI's remarkable capabilities throughout the software development lifecycle. Enhancements in areas such as code generation, code inspection, and software testing have significantly elevated both efficiency and quality. In addition, the potential of AI also provides new possibilities for automated software updates and maintenance in the future. However, despite the broad application prospects of AI in software engineering, it still faces some problems to be solved urgently. For example, inadequate adaptability and the challenges of personal data privacy protection limit its wider application. At the same time, the high research cost and immature model technology also bring obstacles to further development. By comprehensively analyzing the existing literature and related cases, this study deeply discusses the application status and limitations of AI in software engineering. The research results show that although AI can greatly improve the efficiency of software development, its shortcomings in data security and adaptability need attention. Future research should address these problems and seek more effective technical solutions to promote the sustainable development of AI in software engineering.

Keywords: artificial intelligence, software engineering, natural language processing, deep learning

1. Introduction

Artificial intelligence (AI) refers to theories, methods, technologies, and application systems that simulate, extend, as well as enhance human intelligence. And its core goal is to give computers human-like learning and reasoning capabilities, thereby providing solutions based on human perspectives. Since the concept of AI was first put forward at Dartmouth Conference in 1950s [1], after years of in-depth research and technological evolution, AI has become pervasive in a number of industries such as manufacturing, education and transportation, significantly improving production efficiency, optimizing product quality, and providing strong support for the decision-making process [2]. In the manufacturing field, AI can use robots and sensors to reduce human intervention, thus improving production efficiency. In the field of transportation, assisted or autonomous driving can be used to reduce the accident rate and improve the traffic efficiency [3]. Furthermore, in software engineering, AI improves the efficiency of software development by automating software development processes such as code generation and optimization. Also, automated testing and error detection improves the stability and reliability of the developed software [4]. The paper aims to deeply explore the current status of the application of AI in software engineering, analyze the main challenges it faces, and look forward to the future development direction. In order to achieve this goal, this paper adopts the method of literature review, and seeks to provide a comprehensive perspective through the comprehensive analysis of relevant research results and application cases, so as to infer the potential future development possibilities.

2. Application of artificial intelligence in software engineering

2.1. Code generation

The code generation includes data pre-processing, model training, code output and subsequent optimization processing. Data pre-processing, the basis of code generated by AI, is a way to clean, label and format the original data. Through these steps, data that matches the features are extracted to provide data support for subsequent model training. And model training is to build and train code-generated models [5], such as GPT series models. These models learn a lot of information such as syntax, semantics and structure of various codes, and thus complete the generation of codes that meet the requirements. At the same time, by using the natural language understanding (NLU) part of natural language processing (NLP) to process the input information, AI can understand its requirements. The appropriate code template is selected to generate the relevant executable code [6].

AI generated code, also known as intelligent code generation, can be divided into two completely different modes, namely code generation and code completion. And code generation is divided into two parts: one is to generate a large number of cumbersome repetitive codes; the other is that the compiler describes the code requirements to the AI, which generates code with specific requirements through the trained model. However, code completion is completely different. The difference between the two is that the former only inputs the requirements, and the subsequent generation is completed independently by AI. In the latter, the AI predicts the compiler’s intent by understanding the part of the code that has been completed during the compiler's development of the code, and then completes the remaining code [7]. For example, artificial intelligence generated content (AIGC) technology can help developers automatically complete code annotation, completion and other tasks through natural language processing models such as ChatGPT, thus improving coding efficiency. Although the two codes are different in the generation methods, this intelligent code generation process can significantly reduce the manual input time of compilers and speed up the efficiency of code generation [3, 8]. It is worth pointing out that no specialized knowledge is required for users of AI-generated code, markedly lowering the barrier to entry for successful code generation and provides a convenient option for a wider group of users.

2.2. Code inspection

In the code inspection, AI typically relies on machine learning and NLP techniques aimed at analyzing code and identifying issues that do not meet standards [3, 8]. Specifically, a security hole, although it is not an explicit error, poses a potential security risk. Code smell refers to issues that appear innocuous on the surface but may lead to subsequent maintenance difficulties and degrade the quality of the code. The manual identification of these issues is often challenging, as human evaluation is often limited by subjective judgment, thus affecting the efficiency The introduction of AI has significantly improved the effectiveness of this process. Deep learning models enable AI to understand the semantics and structure of code by inputting a large number of code samples. After sufficient training, these models are able to extract code features, giving AI the ability to identify abnormal data [9, 10]. Code inspection is mainly divided into two analysis methods, Static Code Analysis (SCA) and Dynamic Code Analysis (DCA). SCA is usually carried out in the early stages of code development and aims to identify and solve potential problems in time, thereby reducing the cost of later fixes. DCA, on the other hand, performs real-time tests during program running, which can process actual input data and provide accurate information about software performance. Though both methods have their own limitations, for example, SCA is unable to capture runtime errors and DCA is constrained to a particular runtime environment, the combination of these two methods can effectively offset their respective shortcomings, enhancing the comprehensiveness and accuracy of software analysis. After identifying problems in the code, AI can apply automatic code repair technology to locate and resolve the identified errors. The accurate location of the problem is essential for the subsequent generation of effective fix patches. Patches generated by repair tools need to be validated to ensure that they work properly while maintaining original functionality and semantics. For developers, AI code inspection is often more efficient than manual inspection, which not only improves the quality of the code, but also reduces the cost of maintaining the code later. With the continuous advancement of science and technology, future AI is expected to deal with more complex problems and play a greater role in software maintenance and updates.

2.3. Software testing

Software testing is an assessment of whether software meets prescribed criteria by executing software applications, aiming at finding defects and improving software quality. Unlike code inspection that mainly focuses on static analysis of code to identify potential problems, software testing focuses more on verifying whether the overall functionality of the software meets the design requirements. In the context of digital transformation, AI has rapidly changed the software testing industry. Traditional software testing methods rely on manually writing test cases, which is inefficient and susceptible to subjective factors [11]. The advent of AI, particularly machine learning and deep learning, has given rise to a plethora of novel techniques, including automated testing, defect prediction, and performance optimization. Although the performance of AI in software testing still needs to be improved in complex environments, it has great potential in the field of software testing in the future.

Test case refers to writing a set of test inputs, execution conditions and expected results for a specific test target as a way of determining whether the program's path of operation meets expectations or satisfies requirements [11]. Test case generation technology is a method of automating software test cases by using automation tools and methods. For using AI to test software, test case generation technology can greatly improve test efficiency and ensure the breadth and depth of test coverage. AI-based test case generation technology usually uses large language models (LLMs), such as GPT series models, to understand input data and generate corresponding test cases based on the input data. Recent research and applications show that this technology has been explored and practiced in many scenarios. For example, one study has combined the LangChain framework with an existing platform, resulting in fast and stable automatic test case generation. In addition, test case creation projects based on generative AI have been developed, which support the generation of brain map cases based on requirements, and contain UI automation use case recording and playback based on real machine operations, as well as interface automation use cases based on traffic or code generation. As the technology continues to advance and optimize, it is expected that AI-based test case generation will play a more important role in the software testing process in the future [11, 12].

3. Challenges faced

3.1. Technical limitations

Though AI technology has made significant progress in recent years, there are still obvious limitations in its application in software engineering. The difference between natural language and code makes it difficult for existing NLP techniques to accurately understand and generate code, especially in the understanding of different expressions, and NLP still needs to be improved. Improving its performance relies on the introduction of more data and better model development [6]. In addition, code generation tools such as ChatGPT and GitHub Copilot perform well as programming aids, but the code they generate is not always accurate, and there are often cases where it appears to be reasonable but is actually wrong, which affects its reliability. At present, research improves the accuracy of code by introducing automated verification and error detection techniques, such as combining static and dynamic analysis. The innovation of AI in code generation is limited, primarily relying on the reorganization and summarization of existing data, and it is difficult to break through the framework for truly creative thinking. Thus, AI can only be used as an auxiliary tool in software engineering and cannot completely replace human developers [8].

3.2. Data and model issues

The code generation models still need to be improved in its ability to adapt to many different programming languages. Currently, although there are several tools for automatic code generation, most of the generated code is concentrated in two high-level languages, Java and Python. In contrast, the amount and quality of code data for other languages, such as C, are relatively low, mainly because Java and Python are object-oriented languages with rich library and API support, which are easier to interface with natural languages [13-15]. Therefore, building a multilingual code base to support generation in more programming languages may be an effective way to improve model performance. Also, the training of code-generated models relies on a large amount of data, which brings up the issue of data dependency. To address domain-specific code defects, a large amount of relevant data is usually required to train deep learning models. However, these data may be subject to copyright disputes or subjectively biased due to human generation, which may lead to algorithmic discrimination or biased results. To reduce this dependency, models can be optimized by increasing data diversity, continuous learning and updating [13]. In addition, data privacy issues are important challenges in code generation. Training data may contain sensitive information, such as names, ID numbers, or bank accounts, which may lead to data leakage if not properly handled. Therefore, developers need to employ means such as data desensitization or encryption algorithms to ensure the security of private data during model training.

3.3. Integration and adaptation issues

Given that AI typically encompasses a multitude of subsystems and domains, system integration testing is inherently more intricate than module-level testing. Moreover, the technical specifications for integrating numerous modules into a unified functional module are challenging to fulfill. Besides, due to its intricate nature, it is challenging to identify and categorize its constituent parts, which in turn makes it difficult to develop an integrated system that can fully leverage its capabilities. The adaptability challenge of AI in software engineering has also become a difficulty that both of them must face on the road of development. In the face of complex and changing development environments and user requirements in the field of software engineering, the adaptability requirements for AI are greatly increased. AI needs to have the ability to flexibly adapt to different development environments to cope with complex and ever-changing development environments. And being able to handle various programming languages, frameworks and businesses is also a manifestation of the good adaptability of AI in the development process. In the process of software development, AI also needs to be able to process data and give feedback in real time. Then adjust itself according to the progress of the program and changes in the development environment. At the same time, facing the continuous progress and development in the field of software engineering, AI must have the ability of continuous learning. By absorbing new knowledge and new technologies, we continuously optimize our own performance and applicability.

4. Future prospects

4.1. Technological progress and application prospects

The application of AI in software engineering is promoting the progress and innovation of software development methods. Recent research and technological developments show that the role played by AI agents in the entire process of software development and maintenance is becoming more and more critical with the development of technology. They enhance the capacity of large models to address intricate software development and maintenance challenges by augmenting their inference, memory, and planning capabilities. In the contemporary era, AI has the capacity to encompass the entirety of the software development and maintenance process, including, but not limited to, code design, code generation, code inspection, and software testing. Concurrently, it is capable of developing subsequent applications, including those pertaining to fault location, repair, and maintenance.

The use of AI in software engineering is developing rapidly, especially the emergence of LLMs, which has brought new impetus to software development and maintenance. Moreover, AI agents have significantly improved their ability to solve complex software development and maintenance tasks by enhancing their ability to reason, memory, plan and use expansion tools. At present, these agents can cover the entire process of software development and maintenance, including end-to-end software development, maintenance and application of specific links.

In the process automation and intelligent transformation of software development, AI shows great promise for application development. Integrating knowledge from the field of software engineering into agent design and training large base models specialized in software engineering can help improve the effectiveness and efficiency of agent solutions in the future. Even its integration with the field of software engineering has greatly lowered the threshold of this field, adding more possibilities to the development of intelligent development pattern in the future [8].

4.2. Future research directions

Future research directions include establishing more comprehensive evaluation benchmarks and more realistic evaluation data, exploring innovative human-computer collaboration models, and integrating multimodal perception techniques. The development of artificial intelligence is expected to further promote the intelligent transformation of software engineering, and the combination of low-code platforms and large-scale models will promote the development of low-code development, making multimodal human-computer interaction a potential new trend. In addition, large-scale models may drive changes in software development models and organizational structures, bringing about a new software ecosystem. As technology advances, the role of human-computer interaction in the development process will become more important, and the user experience can be further improved by optimizing human-computer interaction. Although AI can significantly improve software development efficiency and application performance, its high cost is also a major challenge. These costs are mainly centered on the design, development, and maintenance of models. The creation, training, and application of AI models require a large investment of resources, usually supported by specialized teams and high-performance computing resources, leading to labor and computing costs as major expenses. Subsequent maintenance, such as model updating and optimization, is also a critical part of ensuring the quality of the application, adding to the long-term cost burden. Research has shown that the trend towards light-weighting and linearization of models can help reduce these costs and drive large-scale applications of AI. By adopting miniaturized and hybrid expert models, more efficient performance and computational efficiency are expected. In the future, as technology continues to evolve, the cost of applying AI in software engineering will be further reduced, driving wider implementation [8, 16].

5. Conclusion

This paper discusses the application and development of AI technology in the field of software engineering. The results show that AI, especially LLMs, significantly improves the efficiency and quality of software development and maintenance, and promotes the improvement of automation and the reduction of human errors. AI demonstrates great potential in code generation, inspection and testing, and plays a positive role in promoting the innovation of software engineering methods. However, although AI technology can reduce development costs, its dependence on large amounts of data and computing resources also increases initial investment and maintenance costs. This paper suffers from a number of limitations, mainly a relatively brief discussion of the limitations of current AI models. In addition, the impact of factors such as ethics and safety in artificial intelligence applications has not been fully considered. Future research should focus on establishing more comprehensive evaluation criteria to assess the effectiveness of AI applications in software engineering, exploring new modes of human-machine collaboration, and considering how to integrate multimodal perception into existing systems. Also, reducing the development and maintenance costs of AI models will be key to ensuring their feasibility and effectiveness in a wider range of practical applications.

References

[1] Wang, C.Y. (2013) Research on the Dartmouth Conference between the United States and the Soviet Union: 1960-1991. Shaanxi Normal University.

[2] Wikipedia. (2024) History of artificial intelligence. https://en.wikipedia.org/wiki/History_of_artificial_intelligence

[3] Nascimento, E.D., Nguyen-Duc, A., Sundbø, I. and Conte, T.U. (2020) Software engineering for artificial intelligence and machine learning software: A systematic literature review.

[4] Zhou, F.S., Wang, L.Z. and Li, X.D. (2019) Automatic Defect Repair and Validation Approach for C/C++ Programs. *Journal of Software*, 30(5): 1243-1255.

[5] Robert Feldt, Francisco G. de Oliveira Neto, and Richard Torkar. Ways of Applying Artificial Intelligence in Software Engineering (Ways of Applying Artificial Intelligence in Software Engineering (arxiv.org))

[6] Feldt, R., de Oliveira Neto, F.G. and Torkar, R. (2018) Ways of Applying Artificial Intelligence in Software Engineering. International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering, 35-41.

[7] Lu, X.C, and Zhou, L.L. (2024) Program Code Defect Detection Based on Natural Language Processing. *Information Recording Materials*, 25 (08): 174-176.

[8] Yang, Z.Z., Chen, S.R, Gao, C.Y., Li, Z.H., Li, G. and Lu, R.C. (2024) Deep Learning Based Code Generation Methods: Literature Review. *Journal of Software*, 35(2): 604-628.
[9] Xie, T. (2018) Intelligent Software Engineering: Synergy Between AI and Software Engineering. In: Feng, X., Müller-Olm, M., Yang, Z. (eds) Dependable Software Engineering. Theories, Tools, and Applications. SETTA 2018. Lecture Notes in Computer Science, 10998.
[10] Azeem, M.I., Palomba, F., Shi, L. and Wang, Q. (2019) Machine learning techniques for code smell detection: A systematic literature review and meta-analysis. *Inf. Softw. Technol*., 108: 115-138.
[11] Gou, Q.W., Dong, Y.W. and Li, Y.M. (2024) Research progress of program synthesis based on deep learning. *Journal of Computer Science*, 1-36.
[12] Wikipedia. (2024) Test case. https://en.wikipedia.org/wiki/Test_case
[13] Zhang, C., Xu, H.C. and Luo, T.J. (2024) Research on the application of artificial intelligence technology in the field of software testing. Shanxi Science and Technology News.
[14] Deng, X., Ye, W., Xie, R. and Zhang, S.K. (2023) Survey of Source Code Bug Detection Based on Deep Learning. *Journal of Software*, 34(2): 625-654.
[15] Shen, K., Huang, K..F., Chen, B.H, et al. (2024) Python third-party library API compatibility problem detection method based on static analysis. *Journal of Software*, 1-26.
[16] Meziane, F. and Vadera, S. (2009) Artificial Intelligence Applications for Improved Software Engineering Development: New Prospects.