Journal of Global Research in Electronics and Communication
Volume 1, No. 12, December, 2025
Available Online at: www.jgrec.info
REVIEWPAPER

AI-Powered Tools in Software Engineering Applications in Code Generation and Quality Assurance

Dr. Pradeep Laxkar
Associate Professor
Department of Computer Science and Engineering
ITM(SLS) University, Vadodara, Gujarat
Dpradeep.laxkar@gmail.com

***Abstract*—Modern software engineering is being red financed by tools based on Artificial Intelligence (AI) and specifically in the realms of the code generation, quality assurance, and continuous integration and deployment (CI/CD). Machine learning, deep learning, natural language processing, and large language model and practice have made intelligent systems shift of traditional rule-based automation to collaborative development support. GitHub Copilot and transformer-based models introduced by Open AI are examples of tools which have shown significant advances in developer productivity, code quality and automated error detection through presented code suggestions and intelligent refactoring options. In quality assurance, AI-based methods enable automated test generation, defect prediction, and anomaly detection, which help to make the process of manual testing much easier and enhance the reliability of the software. In a similar fashion, AI-based CI/CD pipelines make use of predictive analytics and real-time performance inspection to optimize build phases, identify deployment anomalies, and improve system stability. Nonetheless, the implementation of AI in software engineering also brings about issues of data quality, interpretability of model, trust and ethics issues that should be addressed to ensure responsible deployment the increase in the significance of human AI cooperation and the attainment of scalable, dependable, and ethically appropriate inclusion of AI into modern software engineering practice.**

Keywords—Artificial Intelligence, Software Engineering, Code Generation, Quality Assurance, Large Language Models, Machine Learning

I. INTRODUCTION

The dynamic nature of software engineering has been highly affected by the development of Artificial Intelligence (AI) that has shifted the paradigm of software-system development and maintenance. AI-driven collaboration systems [1]. Traditionally, AI applications in software development were limited to rule-based programming assistants or automated code generation models [2][3][4]. However, recent advancements in large language models (LLMs) and multi-agent systems (MAS) have transformed the role of AI from a mere auxiliary tool to an active collaborator in the software engineering process. MAS enables multiple, autonomously handling different aspects of software development, from requirement analysis to debugging and optimization.

AI-augmented code generation extensively in recent years, especially with the rise of deep learning models trained on massive codebases [5]. The tools can also create source code and propose the best code structure by utilizing machine learning models that have been trained on large-scale code repositories, and can help the developer through intelligent code completion and refactoring [6][7]. Rule-based approaches and syntax-driven generation of transformer models, like Open Ai's GPT-based Codex and Google's BERT, has significantly improved AI's ability to understand context and generate human-like code. These models analyse vast repositories of public code, learning patterns, best practices, and common bugs, could reduce code-writing effort by up to 30% [8]. AI-assisted coding accelerates code development, especially in repetitive or boilerplate code and can even enhance the quality of the code, such as by proposing optimization.

AI-optimized code has become a new ground-breaking strategy to deal with those issues [9][10]. The AI-based tools can scan the pattern of a code, identify its inefficiency and suggest or automatically make changes, using machine learning, deep learning and reinforcement learning tools. [11]. These tools enhance the efficiency of CI pipelines by identifying performance bottlenecks, predicting potential failures, and ensuring that code adheres to best practices. Despite its promise,

AI-powered code optimization faces several challenges, including the need for high-quality training data, the integration of AI models with existing CI workflows, and concerns regarding interpretability and trust [12]. It delves into the mechanisms of AI-powered code the improvement of the quality of codes and reliability of the systems [13][14]. With the further evolution of AI technologies, the use of AI will remake the software engineering standards, and AI-driven development of software systems.

II. AI-BASED CODE GENERATION

Code generation is an automated process that converts structured or unstructured input information (such as natural language requirements descriptions, design documents, code snippets, etc.) into source code (see Figure 1). Its essence is to reflect the abstract intentions and task goals of the developers into specific programming projects. And based on LLM (Larg e Language Model) for code generation, by breaking down the tasks, having data storage with long-term and short-term memories, as well as the invocation of external tools, these are currently important technical supports in the field of code generation[15]. the application effects and code generation quality of Codex and Co-pilot in the field of code generation.

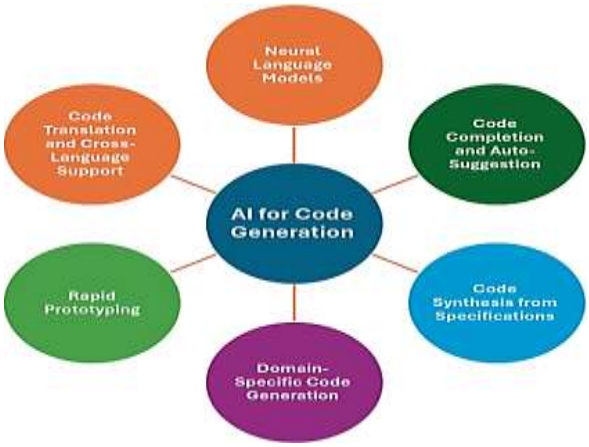

Fig. 1. AI -based code generation

A. *Types of AI Code Generation Tools*

The AI code generators may be classified according to the functionality and the automation degree they offer in the software development life cycle in Figure 2. They are tools that rely on methods like machine learning, deep learning, and natural language processing to help developers write, refine, and maintain code effectively.

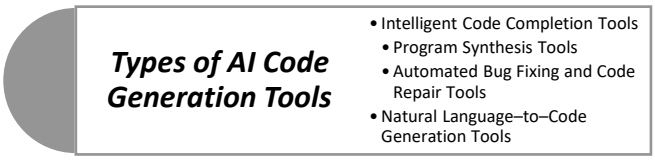

Fig. 2. Types of AI code generation tools

1) *Intelligent Code Completion Tools*

These tools can help the developers as they can predict and propose the code snippets, functions or even complete line of code as they type in real time [16]. Intelligent code completion tools can prevent syntax errors, increase coding productivity and enforce compliance with programming by learning syntax from existing large codebases and learning behave by studying the behavior of developers. They are frequently incorporated with current integrated development environments (IDEs) the productivity of the developer.

2) *Natural Language–to–Code Generation Tools*

Natural language to code translators enables the description by developers of the functionality in human language, which is then converted into runnable source code. The tools close the requirements implementation gap and allow quicker prototyping and reduced the threshold of the non-expert programmers. They are specifically handy when it comes to generating boilerplate code, APIs and simple application logic.

3) *Program Synthesis Tools*

Program synthesis tools are computer programs that use formal specifications, constraints, or example inputs and outputs to automatically generate complete programs or code segments [17]. These tools may generate correct-by-design code through reasoning over definitions of problems and are useful in areas of high reliability.

4) *Automated Bug Fixing and Code Repair Tools*

The tools are aimed at identifying the errors and automatically creating remedies to bugs in the source code. AI-based code repair systems can use pattern recognition and historical bug-fix data to propose patches or corrections, dramatically cutting debugging time and enhancing software reliability.

B. *Automated Code Synthesis and Completion*

Traditional code completion methods usually depend on type information generated during compilation to predict the next token. Since in static languages, types are fixed at compile time, and type information is key information for code completion, this method shows good Performance in statically typed programming languages such as Java [18]. However, in dynamic programming languages (e.g., Python, JavaScript), the type of a variable is determined at runtime and can change as the program executes. To effectively solve this problem, researchers began to use the naturalness of code for code completion.

C. *Program Translation and Refactoring*

Refactoring operation is defined as the process of changing a software system in such a way that it does not alter the function of the code, yet improves its internal structure. It is the art of modifying the design of a system without altering its behavior, with the fundamental concept of behavior preservation being central to refactoring operation. This process is used to improve system maintainability and extend its usable lifespan [19]. Refactoring operation is applied to restructure design, eliminate, replace, or rewrite code to improve its efficiency and understandability, or to transform applications to use modern infrastructure support functions. Refactoring operation improves design structure while preserving the external behavior, and is one of the most used techniques to ease software maintenance activities such as adding new functionalities, correcting bugs, and modifying code to improve quality.

D. *Low-Code and No-Code Development Platforms*

Low-code and no-code (LCNC) development platforms are software development tools that enable users to build applications with minimal or no coding experience. Low-code platforms provide a visual development environment with drag-and-drop components, pre-built templates, and limited coding capabilities for customization [20]. No-code platforms take this further by eliminating the need for coding, allowing users to create applications through graphical interfaces and logic-based workflows. These platforms democratize application development, making it accessible to non-technical users while also enhancing efficiency for professional developers.

Key Features of LCNC Platforms

- **Drag-and-Drop Interfaces:** LCNC platforms offer intuitive visual development environments, allowing users to design applications by dragging and dropping elements without writing code.
- **Pre-Built Templates and Components:** These platforms provide ready-made templates and reusable components, enabling faster development and reducing the need for custom coding.
- **API and Third-Party Integrations:** LCNC tools support seamless integration with external services, databases, and APIs, enhancing functionality and interoperability with existing systems.
- Automation and Workflow Management: Built-in automation features allow users to create workflows,

trigger actions, and streamline business processes without manual intervention.

III. AI POWERED TOOL IN SOFTWARE ENGINEERING

Artificial intelligence (AI) has caused basic changes in different aspects of software creation. The combination of AI-based tools and techniques has increased the efficiency, precision, and flexibility of the software engineering procedures [21]. This section examines the principal applications of AI within software engineering, illustrating how they revolutionize conventional techniques and augment overall efficiency.

Selection Criteria for Tools and Datasets: The tools, datasets, and examples included in this were selected based on the following criteria:

- **Relevance to Industry Applications:** Tools and datasets that are widely used in real-world software engineering contexts, such as GitHub Co-pilot and IBM's defect prediction tools, were prioritized.
- **Recency:** Preference was given to tools and datasets published or actively used within the past five years to ensure that the study reflects the current state of the field.
- **Accessibility:** Open-source datasets and tools with publicly available documentation were selected to facilitate reproducibility.
- **Coverage of Development Phases:** cover diverse phases of the software development lifecycle, including coding, testing, and maintenance [22].
- Impact and Adoption: The selection emphasized tools and datasets with demonstrated effectiveness, as reported in industry and academic studies.

A. Machine Learning and Deep Learning Techniques

Machine learning (ML) and deep learning (DL) have greatly impacted many areas by providing sophisticated technique for data analysis, prediction, and automation [23]. These technologies have played a major role in the development of artificial intelligence (AI) innovations and influenced industries like healthcare, finance, and manufacturing. The methods and techniques in machine learning and deep learning are presented in Table I.

TABLE I. METHOD AND TECHNIQUE IN MACHINE LEARNING PREDICTION IN AI TOOL IN SOFTWARE ENGINEERING

Sr.No	Category	Method/Technique	Description	Applications
1.	Supervised Learning	Linear Regression	A statistical method to model and analyse the relationship between a dependent variable and one or more independent variables.	Predictive analytics, trend forecasting, financial modelling
		Logistic Regression	A classification technique used to predict the probability of a binary outcome based on one or more predictor variables.	Medical diagnosis, fraud detection
		Decision Trees	A tree-structured model is used to make decisions and predict outcomes by splitting data into branches based on feature values.	Risk management, classification tasks.
2.	Un supervised learning	K-Means Clustering	A clustering technique that partitions data into k clusters, where each data point belongs to the cluster with the nearest mean.	Customer segmentation, image compression
		Hierarchical Clustering	A clustering method that builds a hierarchy of clusters by either merging or splitting existing clusters.	Social network analysis, genomic data analysis
		Principal Component Analysis (PCA)	A dimensionality reduction technique that transforms data into a set of uncorrelated variables, called principal components, ordered by the amount of variance they capture.	Data visualization, noise reduction
3.	Reinforcement Learning	Q-Learning	A model-free reinforcement learning An algorithm that learns the value of an action in a particular state by using a policy that maximizes cumulative reward.	Robotics, game playing
		Deep Q-Networks (DQN)	A combination of Q-Learning with deep neural networks, allowing the handling of high-dimensional sensory inputs.	Autonomous vehicles, gaming
4.	Deep Learning	Convolutional Neural Networks (CNN)	A class of deep learning models primarily used for processing grid-like data structures such as images by learning spatial hierarchies of features.	Image and video recognition, medical image analysis
		Recurrent Neural Networks (RNN)	A class of neural networks designed for sequence data, were connections between Nodes form a directed graph along a temporal sequence.	Time series analysis, language modelling, speech recognition

Deep learning models: Deep learning, which is a branch of machine learning, is the training of large neural networks with multiple layers to learn intricate data representations. Convolutional neural networks (CNNs) are leading in the field of image and video processing because they capture spatial hierarchies [24]. Recently, there have been architectures such as Efficient Net, maximising the performance with fewer parameters, and Vision Transformers (ViTs), applying transformer models to image data to achieve improved accuracy and scalability. Neural Networks, Recurrent Neural Network, Recurrent Neural Networks (RNNs), as well as their variations: Long Short-Term Memory (LSTM) networks and Gated Recurrent Units (GRUs), are important in the processing of sequential data, such as language modelling and time-series prediction.

1) Explainable AI

As machine learning models, especially deep learning models, continue to grow more complex, it has been necessary to gain insight into how they make decisions. Explainable AI

(XAI) is dedicated to the goal of making predictions made by models more understandable to humans. Individual prediction explanations. Techniques such as LIME (Local Interpretable Model-agnostic Explanations), SHAP (Shapley Additive explanations) and integrated gradients can be applied to explain a single prediction and learn about model behavior.

2) Federated Learning

Federated learning is a new method that allows the training of models with decentralized devices/servers but local data. The strategy ensures privacy and security because it does not require the concentration of sensitive information. Federated learning also comes in handy in those industries, where data privacy is paramount, including healthcare and finance. Recent studies in federated learning solve such issues as the effectiveness of communication, heterogeneity of data, and safety of aggregation to make the application robust and scalable.

B. Natural Language Processing for Software Tasks

NLP-based software testing is faced with numerous challenges and open issues that require proper consideration and creativity. To begin with, natural language is ambiguous and variable, which is a major challenge to the proper interpretation and analysis of textual artefacts, including requirements documents, user stories, and software specifications. More so, software engineering is domain-specific, which presents even more complexities, and NLP models need to be customizable and specialized to accommodate various linguistic patterns and terms [25]. Moreover, the interpretability and trustworthiness of NLP models in the context of software testing raise profound concerns regarding reliability, robustness, and ethical considerations [26]. It provides the current state of a certain subject by using rigorous and analysis in Figure 3.

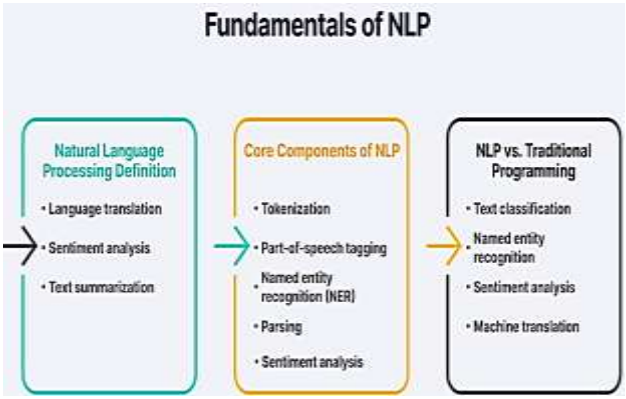

Fig. 3. Fundamentals of NLP

Also, the scalability and generalizability of NLP-based testing frameworks remain elusive goals, with existing approaches often struggling to cope with the complexities of large-scale software systems and diverse testing scenarios.

C. Large Language Models in Programming

Large Language Models (LLMs) have shown notable performance in generating source code, acting as development bots (DevBots) to enable human-bot collaboration in software projects. These models perform effectively in practical downstream tasks such as generating code from natural language descriptions [27]. These advancements in LLM for complex code generation have facilitated developers with increased automation and enhanced the role of such models in software development. This model completes the full software development process in less than seven minutes at a cost of under one dollar. It introduced Codegrees, a multilingual model with 13 billion parameters, specifically designed for code generation across 23 programming languages [28]. Metate, a metaprogramming framework that integrates efficient human workflows into LLM-based multi-agent collaborations.

IV. Continuous Integration And Deployment For Software Quality Assurance

Continuous Integration (CI) is a software development practice where developers frequently integrate their code changes into a shared repository, typically multiple times a day. Each integration triggers an automated build and testing process, allowing teams to detect and address issues early in the development cycle. CI emphasizes collaboration, encouraging developers to work together to produce a stable codebase [29]. Continuous Deployment (CD) extends the principles of CI by automating the release of code changes to production environments after passing predefined testing criteria. In a CD pipeline, successful builds are automatically deployed, enabling teams to deliver new features and fixes to users rapidly and consistently. CI and CD create a seamless workflow that enhances software delivery, ensuring that updates can be made quickly and safely with minimal manual intervention.

A. Benefits of CI/CD

Implementing CI/CD practices offers numerous benefits that contribute to more efficient and effective software development:

1) Faster Release Cycles

One of the most significant advantages of CI/CD is the acceleration of release cycles. By automating the integration and deployment processes, teams can deliver new features, enhancements, and bug fixes more frequently. This agility allows organizations to respond rapidly to market demands and user feedback, ensuring that their software remains competitive and relevant.

2) Improved Code Quality

CI/CD lead to enhanced code quality through rigorous automated testing and continuous feedback. By integrating testing into the development pipeline, teams can identify and resolve defects early in the process, reducing the likelihood of issues in production. The emphasis on frequent integration also encourages developers to adhere to coding standards and best practices, further contributing to the overall quality of the codebase.

3) Enhanced Collaboration Among Teams

CI/CD fosters a collaborative environment among developers, testers, and operations teams. By promoting shared responsibilities and clear visibility into the development process, CI/CD reduces silos and encourages open communication. Teams are more likely to work together to solve problems, share knowledge, and support one another, ultimately leading to a more cohesive and productive development culture.

B. Intelligent Build and Deployment Pipelines

A Deployment pipeline is the process of taking code from version control and making it readily available to users of your application in an automated fashion [30]. When a team of developers are working on projects or features, they need a

reliable and efficient way to build, test and deploy their work. Historically, this would have been a manual process involving lots of communication and a lot of human error.

The stages of a typical deployment pipeline are as follows in Figure 4.

Fig. 4. Deployment pipeline

One of the primary benefits of AI in CI/CD is its ability to detect errors early in the development cycle. Traditional CI/CD tools rely on rule-based systems that may not catch subtle or complex bugs. AI enhances error detection through:

1) Predictive Analytics

AI models analyze historical data to predict potential failures before they occur [31]. By identifying patterns in past deployments, predictive analytics help developers pre-emptively address issues.

2) Anomaly Detection

Machine learning algorithms monitor logs, metrics, and test results to detect anomalies. These algorithms flag deviations from normal behavior, enabling rapid debugging and resolution.

3) Automated Code Review

AI-driven tools review code submissions for syntax errors, security vulnerabilities, and inefficiencies. This reduces manual review time and improves code quality before deployment.

C. AI-Based Performance Monitoring

Real-time performance monitoring for artificial intelligence components in interactive applications has emerged as a critical discipline in modern computing. As users engage with intelligent systems through voice assistants, adaptive interfaces, predictive analytics dashboards, and autonomous decision agents, the need to understand how these systems perform under dynamic conditions intensifies. Traditional software monitoring focuses on static metrics such as uptime, throughput, and error rates [32]. Real-time AI performance monitoring seeks to capture these multi-faceted performance indicators as systems operate, enabling developers, operators, and stakeholders to make informed decisions that improve responsiveness and trustworthiness.

V. LITERATURE REVIEW

The literature review that Artificial Intelligence has a software engineering AI-powered techniques coding, testing, automation, and system optimization processes efficient improving productivity, quality, and reliability in Table II of focus area, key finding, challenges and future work are discussed below:

Kosna (2025) the integration of Artificial Intelligence (AI) into software development has triggered a paradigm shift, fundamentally reshaping the landscape of software engineering the transformative impact of AI across the entire Software Development Lifecycle (SDLC), from requirements engineering to deployment and maintenance. We explore the role of Generative AI (GenAI) in code generation and automation, the advancements in AI-driven software testing and quality assurance, and the evolution of DevOps through intelligent CI/CD pipelines [33].

Peterson, Benjamin and Johnson (2025) artificial Intelligence (AI) has emerged as a transformative force in software engineering, reshaping the development lifecycle from initial code generation to deployment. The evolution and integration of AI-powered automation within software engineering processes, innovations in natural language-based coding, intelligent testing, continuous integration, and deployment strategies scholarly and industry sources [34].

Chen et al. (2024) the rapid advancements in Generative AI (GenAI) tools, such and GitHub Copilot, are transforming software engineering by automating code generation tasks. While these tools improve developer productivity, for organizations and hiring professionals in evaluating software engineering candidates' true abilities and potential these tools in both industry and academia, tools specifically affect the hiring process [35].

Sajid and Maya (2023)AI-powered software engineering is transforming the way software development processes are managed, particularly in the area of code generation. Traditionally, software engineering has been a highly manual and time-consuming process, with developers needing to write large amounts of code, troubleshoot bugs, and handle complex requirements. Multi-agent systems (MAS) leverage multiple AI agents working collaboratively to perform tasks typically require human intervention software engineering, these agents can automate code generation, testing, debugging, and even management [36].

Pham, Nguyen and Nguyen (2022) state that software testing is a process of evaluating and verifying whether a software product still works as expected, and it is repetitive, laborious, and time-consuming automation tools have been developed to automate testing activities and enhance quality and delivery time. Recent advances in artificial intelligence and machine learning (AI/ML) the potential for addressing important in test automation applied to automate various testing activities, such as detecting bugs and errors, maintaining test cases, or generating new test cases much faster than humans [37].

Mulla and Jayakumar (2021) artificial intelligence (AI) and machine learning (ML) techniques in the field of software testing. The use of AI in software testing is still in its initial stages. Also, the automation level is lower compared to more evolved areas of work. AI and ML can be used to help reduce tediousness and automate tasks in software testing the potential of between human and machine-driven testing capabilities to fully utilize AI and ML techniques in testing, enhance the entire testing process and skills of testers and will contribute to business growth [38]

TABLE II. COMPARATIVE ANALYSIS OF AI POWERED TOOL IN SOFTWARE ENGINEERING IN CODE GENERATION

Author	Focus Area	Key Findings	Approach	Challenges	Future Work
Kosna (2025)	AI integration across SDLC	AI has caused a paradigm shift in software engineering by transforming all SDLC phases, including requirements, development, testing, deployment, and maintenance.	Conceptual and analytical review of AI applications across SDLC with emphasis on GenAI and DevOps	Integration complexity, reliability of AI-generated artifacts, and governance of AI-driven pipelines	Deeper empirical validation of AI-driven SDLC tools and development of standardized AI governance frameworks
Peterson, Benjamin & Johnson (2025)	AI-powered automation in software engineering	AI reshapes the development lifecycle through intelligent code generation, testing, CI/CD, and deployment strategies, increasing efficiency and adaptability.	Systematic review of scholarly and industry literature	Trust in AI-generated code, explainability, and alignment with existing workflows	human AI collaboration models and explainable AI in software engineering
Chen et al. (2024)	Generative AI tools in code generation	Tools like ChatGPT and GitHub Copilot significantly improve productivity but raise concerns in evaluating developers' real skills, especially.	Empirical analysis and industry-academia comparison	Skill assessment challenges, over-reliance on AI tools, academic integrity issues	Development of new evaluation frameworks for hiring and education in the presence of GenAI
Sajid & Maya (2023)	Multi-Agent Systems (MAS) for AI-powered software engineering	MAS enable collaborative AI agents to automate code generation, testing, debugging, and project management, reducing manual effort.	Architecture-based analysis of MAS in software engineering	Coordination overhead, system complexity, and scalability issues	Optimization of agent coordination and real-world deployment of MAS-based frameworks
Pham, Nguyen & Nguyen (2022)	AI/ML in software test automation	AI/ML significantly accelerate testing by automating bug detection, test case generation, and maintenance, quality and delivery speed.	AI/ML techniques applied to software testing	Data dependency, test reliability, and tool integration issues	Advanced learning models for adaptive and self-maintaining test systems
Mulla & Jayakumar (2021)	AI and ML in software testing	AI/ML adoption in testing is still immature but shows strong potential to reduce manual effort	Exploratory of AI/ML techniques in testing	Low automation maturity, lack of skilled testers, limited real-world adoption	Skill development for testers and research on hybrid human–AI testing frameworks

VI. CONCLUSION WITH FUTURE WORK

The transformative nature of Artificial Intelligence in current software engineering, specifically, AI-assisted code generation, quality assurance, and CI/CD practices. The recent development in machine learning, deep learning, natural language processing, and large language models has transformed the concept of AI from an automation based on rules into a partner in development. The productivity of the developer, plus the quality of the code, and the detection of the defect can be seen in tools like GitHub Copilot and transformer-based models created by Open AI, which have proven to have made significant improvements. The paper has pointed out the role of intelligent build pipelines, predictive analytics, anomaly detection, and real-time performance monitoring to improve reliability and speed in the delivery of software process of data quality dependency, low levels of interpretability, trust, ethical issues, and excessive dependence on automation. Development practices that integrate, transparently, and in a governable manner. Future studies must be directed towards justifiable and credible AI frameworks for CI/CD pipelines, evaluation benchmarks, and ethical governance constructs. Also, cross-domain discovery, like the implementation of AI-based optimization ideas to expansion loops of pressure piping systems, provides encouraging possibilities of helping to enhance structural stability and design safety.

REFERENCES

[1] J. Sanchez, A. Willie, and M. Niall, "AI-Powered Code Optimization in Continuous Integration," 2024.

[2] P. Gupta, S. Kashiramka, and S. Barman, "A Practical Guide for Ethical AI Product Development," in *2024 IEEE 11th Uttar Pradesh Section International Conference on Electrical, Electronics and Computer Engineering (UPCON)*, 2024, pp. 1–6. doi: 10.1109/UPCON62832.2024.10983504.

[3] S. Kashiramka, P. Gupta, and S. Barman, "Literature Review of Artificial Intelligence's Impact on Politics and Society," in *2024 IEEE 11th Uttar Pradesh Section International Conference on Electrical, Electronics and Computer Engineering (UPCON)*, 2024, pp. 1–6. doi: 10.1109/UPCON62832.2024.10983037.

[4] K. S. Hebbar, "AI-Driven Code Review: A Real-Time Feedback System for Secure and Maintainable Software Development," *J. Inf. Syst. Eng. Manag.*, 2024.

[5] R. Wang, R. Cheng, D. Ford, and T. Zimmermann, "Investigating and Designing for Trust in AI-powered Code Generation Tools," 2023. doi: 10.48550/arXiv.2305.11248.

[6] P. R. Marapatla, "Intelligent APIs: AI-Powered Ecosystem for Nonprofit Digital Transformation," *J. Inf. Syst. Eng. Manag.*, vol. 10, 2025.

[7] P. Chandrashekar, "A Survey of Tools, Techniques, and Best Practices: CI/CD Integration in DevOps Workflows," *Int. J. Adv. Res. Sci. Commun. Technol.*, vol. 3, no. 3, pp. 1366–1376, Jul. 2023, doi: 10.48175/IJARSCT-11978V.

[8] J. Erizo, "AI-Augmented Code Generation," *J. Sist. Inf. dan Tek. Inform.*, vol. 3, pp. 19–24, 2025, doi: 10.70356/jafotik.v3i1.53.

[9] P. Chandrashekar, "Enhancing Software Application Efficiency Through Design- Centric Methodologies : An Empirical Evaluation," vol. 2, no. 1, pp. 187–196, 2022, doi: 10.56472/25832646/JETA-V2I1P122.

[10] H. P. Kapadia, "AI Enhanced Web Accessibility Features," vol. 8, no. 4, pp. 476–483, 2021.

[11] W. Nasir and N. Kallinteris, "From Code Generation to AI Collaboration: The Role of Multi-Agent Systems in Software Engineering," 2025. doi: 10.13140/RG.2.2.21102.32320.

[12] S. Roobini, M. Kavitha, H. Deenadayalan, and A. Muthusamy, "AI-Powered Tools to Enhance the Stages of Software Development," 2025, pp. 435–478. doi: 10.4018/979-8-3693-9356-7.ch017.

[13] S. K. Chintagunta, "Generative AI Approaches to Automated Unit Test Case Generation in Large-Scale Software Projects," *ESP J. Eng. Technol. Adv.*, vol. 4, no. 1, pp. 150–157, 2024, doi: 10.56472/25832646/JETA-V4I1P121.

[14] S. K. Chintagunta and S. Amrale, "AI in Code , Testing , and Deployment : A Survey on Productivity Enhancement in Modern Software Engineering," *Int. J. Res. Anal. Rev.*, vol. 10, no. 4, pp.

747–752, 2023, doi: 10.14741/ijcet/v.13.6.16.

[15] Y. Wang, "A Review of Research on AI-Assisted Code Generation and AI-Driven Code Review," *Acad. J. Sci. Technol.*, vol. 18, pp. 236–241, 2025, doi: 10.54097/d6775287.

[16] T. Kaluarachchi, "From Mock-ups to Code: A Conceptual Synthesis of AI-Driven Automatic Website Generation," *Int. J. Innov. Sci. Res. Technol.*, pp. 305–317, Nov. 2025, doi: 10.38124/ijisrt/25nov339.

[17] V. Rajendran, D. Besiahgari, S. C. Patil, M. Chandrashekaraiah, and V. Challagulla, "A Multi-Agent LLM Environment for Software Design and Refactoring: A Conceptual Framework," in *SoutheastCon 2025*, IEEE, Mar. 2025, pp. 488–493. doi: 10.1109/SoutheastCon56624.2025.10971563.

[18] Y. Yin, J. Liu, Y. Liu, and J. Deng, "Advancing code completion through rotary position embedding," *Cluster Comput.*, vol. 28, 2025, doi: 10.1007/s10586-024-05015-z.

[19] A. Ouni, M. Kessentini, and H. Sahraoui, "Multiobjective Optimization for Software Refactoring and Evolution," vol. 94, 2014, pp. 103–167. doi: 10.1016/B978-0-12-800161-5.00004-9.

[20] F. Chad, "Low-Code and No-Code Development Platforms," 2025.

[21] M. Alenezi and M. Akour, "AI-Driven Innovations in Software Engineering: A Review of Current Practices and Future Directions," *Appl. Sci.*, vol. 15, no. 3, p. 1344, Jan. 2025, doi: 10.3390/app15031344.

[22] S. Martínez-Fernández *et al.*, "Software Engineering for AI-Based Systems: A Survey," *ACM Trans. Softw. Eng. Methodol.*, vol. 31, pp. 1–59, 2022, doi: 10.1145/3487043.

[23] D. Patil, N. Rane, P. Desai, and J. Rane, "Machine learning and deep learning: Methods, techniques, applications, challenges, and future research opportunities," 2024, pp. 28–81. doi: 10.70593/978-81-981367-4-9_2.

[24] M. Soori, B. Arezoo, and R. Dastres, "Artificial intelligence, machine learning and deep learning in advanced robotics, a review," *Cogn. Robot.*, vol. 3, pp. 54–70, 2023, doi: 10.1016/j.cogr.2023.04.001.

[25] M. Boukhlif, M. Hanine, N. Kharmoum, A. Noriega, D. Obeso, and I. Ashraf, "Natural Language Processing-Based Software Testing: A Systematic Literature Review," *IEEE Access*, vol. PP, p. 1, 2024, doi: 10.1109/ACCESS.2024.3407753.

[26] Z. Pauzi and A. Capiluppi, "Applications of natural language processing in software traceability: A systematic mapping study," *J. Syst. Softw.*, vol. 198, p. 111616, Apr. 2023, doi: 10.1016/j.jss.2023.111616.

[27] R. A. Husein, H. Aburajouh, and C. Catal, "Large language models for code completion: A systematic literature review," *Comput. Stand. Interfaces*, vol. 92, p. 103917, Mar. 2025, doi: 10.1016/j.csi.2024.103917.

[28] Z. Rasheed *et al.*, "Large Language Models for Code Generation: The Practitioners Perspective," 2025. doi: 10.48550/arXiv.2501.16998.

[29] H. Agoro and O. James, "AI-Enhanced Continuous Integration and Deployment (CI/CD)," 2022.

[30] D. Merron, "Deployment Pipelines (CI/CD) in Software Engineering," *https://www.bmc.com*, 2020.

[31] S. Chukwueze, A. Akano, D. David, and A. Samuel, "AI-Driven Evolution of CI/CD Pipelines: Intelligent Error Detection and Performance Optimization," 2025.

[32] J. Mitchell, R. Carter, M. Hughes, O. Reynolds, and D. Esther, "Real-Time AI Performance Monitoring in Interactive Applications," 2025.

[33] S. R. Kosna, "AI-Driven Software Development: A Paradigm Shift in Engineering Practices," 2025.

[34] E. Peterson, M. Benjamin, and E. Johnson, "AI-Powered Automation in Software Engineering: From Code Generation to Deployment," 2025.

[35] A. Chen, T. Huo, Y. Nam, D. Port, and A. Peruma, "The Impact of Generative AI-Powered Code Generation Tools on Software Engineer Hiring: Recruiters' Experiences, Perceptions, and Strategies," 2024. doi: 10.48550/arXiv.2409.00875.

[36] B. Sajid and K. Maya, "AI-Powered Software Engineering: Automating Code Generation with Multi-Agent Systems," 2023. doi: 10.13140/RG.2.2.15754.58560.

[37] P. Pham, V. Nguyen, and T. Nguyen, "A review of ai-augmented end-to-end test automation tools," in *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*, 2022, pp. 1–4.

[38] N. Mulla and N. Jayakumar, "Role of Machine Learning \& Artificial Intelligence Techniques in Software Testing," *Turkish J. Comput. Math. Educ.*, vol. 12, no. 6, pp. 2913–2921, 2021..

