ISSN (Online): 2455-3662
EPRA International Journal of Multidisciplinary Research (IJMR) - Peer Reviewed Journal
Volume: 11| Issue: 11| November 2025|| Journal DOI: 10.36713/epra2013 || SJIF Impact Factor 2025: 8.691 || ISI Value: 1.188

GENERATIVE AI: TRANSFORMING SOFTWARE ENGINEERING PRACTICES

Lakki Ali[1], Dr. Archana Kumar[2]
[1]*Department of Artificial Intelligence and Data Science, Dr. Akhilesh Das Gupta Institute of Professional Studies*
[2]*Professor, Department of Artificial Intelligence and Data Science, Dr. Akhilesh Das Gupta Institute Of Professional Studies*

ABSTRACT

*Generative Artificial Intelligence (GAI) is fundamentally changing **how software is created**, acting as a powerful new **co-pilot** for developers. It leverages large language models (LLMs) to automate essential tasks like **writing code, fixing bugs, and improving performance**, offering developers **real-time suggestions** and automatically catching errors. This has led to huge efficiency gains, potentially **reducing development time by significant margins**. However, this integration presents challenges that must be addressed, notably ensuring the **accuracy and reliability** of the generated code, overcoming the AI's current limitations in understanding **complex project requirements**, and mitigating pressing **security and ethical concerns** related to intellectual property and code privacy. The path forward demands continuous **model improvement**, integrating **secure AI practices**, and fostering a deep level of **Human-AI collaboration** to augment developer capabilities, ultimately redefining programming by prioritizing the creation of **accurate, reliable, and ethically sound** software environments.*

1. INTRODUCTION

Generative Artificial Intelligence (**GAI**) is fundamentally shaking up software engineering, giving developers powerful, **AI-driven assistance** that works alongside traditional coding methods. This is a big deal. Unlike older forms of AI that just predict or categorize things, GAI uses sophisticated models—especially **Large Language Models (LLMs)**—to actually **create new, useful content** like functional code, documentation, and design ideas. This ability to *generate* content means GAI can automate tasks across the entire development process.

Tools like **GitHub Copilot** and **Amazon CodeGuru** act as smart assistants, offering **real-time code suggestions**, completing complex snippets, and even writing entire functions just from a simple text description. By providing this intelligent help and automatically catching errors and potential security issues, GAI doesn't just make development faster (some reports show potential **time savings up to 45%**); it also helps improve the **quality and security** of the final code.

However, bringing GAI into the workplace isn't without its challenges. Developers have to be careful because the AI-generated code isn't always **100% accurate or reliable**, meaning human review is essential. The tools can also struggle with **understanding the bigger picture**, sometimes missing the specific business logic or complex project context. Most importantly, integrating GAI raises significant **security, privacy, and ethical concerns**—we have to protect proprietary code, prevent the generation of insecure code, and address worries about intellectual property and how this might change jobs.

To get the most out of GAI, we need to carefully look at what these tools can do now and where they fall short. The future lies in making the models smarter, giving them better contextual awareness, developing **secure ways to train them**, and establishing clear **ethical guidelines** to ensure GAI enhances human capabilities and helps us build software that is accurate, trustworthy, and collaboratively built.

The role of Artificial Intelligence (**AI**) in software development has gone through a huge shift, moving from tools that simply **predict and categorize** to the revolutionary ability of **Generative AI (GAI)** to autonomously **create new content**.

2. LITERATURE REVIEW

Generative Artificial Intelligence (GAI) is dramatically changing how we build software. It's not just an incremental step; it's a complete **transformation**. Instead of just predicting outcomes, like older Machine Learning (ML) does, GAI uses sophisticated models—especially **Large Language Models (LLMs)** like the GPT series (with big leaps in 2020, 2021, and 2023)—to actively **create new, meaningful content**, such as code, design mockups, and documentation. This power slashes coding time, catches bugs, and boosts everyone's output.

2.1 Writing and Finishing Code

Tools like **GitHub Copilot** are true coding co-pilots. They can translate your simple English description into working code, generate common, repetitive code, and finish your lines in real-time. This greatly reduces the mental effort needed to code. Studies back this up, showing GAI can cut development time by as much as **45%**. Some

ISSN (Online): 2455-3662
EPRA International Journal of Multidisciplinary Research (IJMR) - Peer Reviewed Journal
Volume: 11| Issue: 11| November 2025|| Journal DOI: 10.36713/epra2013 || SJIF Impact Factor 2025: 8.691 || ISI Value: 1.188

Copilot users even report that the tool auto-generates and tests nearly half their code, speeding up their workflow by **55%**.

2.2 Finding and Fixing Bugs

GAI is a powerful quality controller. Tools like **DeepCode** and **Amazon CodeGuru** constantly scan codebases to instantly spot errors, security holes, and slow performance spots, offering smart suggestions on how to fix them. The AI can automatically write test cases and predict potential issues, saving huge amounts of time in testing and maintenance. For example, one AI framework showed an impressive bug-spotting accuracy of **91.4%**.

2.3 Making Code Easy to Manage

GAI helps ensure code stays clean. It checks that developers are following established standards, automatically writes **comprehensive documentation and comments** based on the code's structure, and even suggests better ways to organize the code (**refactoring**). Automated documentation is a huge time-saver and ensures the instructions always match the current code.

2.4. Speed and Quick Prototypes

By automating the simple, repetitive chores, GAI frees developers to focus on the truly complex and creative features. It also dramatically speeds up the early stages of a project by quickly generating working models (**functional prototypes**) just from a high-level idea or a natural language request.

2.2.1 POPULAR AI CODING TOOLS

The modern software development landscape is significantly enhanced by several popular **AI-powered coding tools**, each leveraging unique technology to streamline workflows. **GitHub Copilot**, powered by **OpenAI Codex**, is perhaps the most well-known, excelling at providing **real-time code suggestions and basic generation**. It's primarily suited for general software development and rapid prototyping, essentially acting as a tireless pair programmer. Similarly, **OpenAI Codex** itself is best for multi-language projects and experimentation, as its core strength lies in **translating natural language descriptions into code**. For developers prioritizing code quality and security, **DeepCode** utilizes custom AI models to offer **real-time code analysis and smart fix suggestions**, making it ideal for security-sensitive applications. **Amazon CodeGuru**, built on **AWS ML models**, targets larger corporate environments by offering **automated code reviews** and finding critical security problems and performance bottlenecks. Finally, **IntelliCode** uses models trained on vast GitHub projects to provide **context-specific advice based on industry best practices**, helping ensure the final code adheres to high standards and quality. Together, these tools automate critical, time-consuming tasks, allowing developers to focus on complex, creative problem-solving.

2.2.2 THE ROADBLOCKS AND WHAT'S NEXT

Despite the clear benefits, using GAI comes with serious hurdles we need to address:

- **Is the Code Right? (Accuracy and Reliability)**The AI can sometimes write incorrect or poorly performing code. Developers *must* review and test everything. Future efforts need to focus on training models better, using **hybrid systems** (AI plus traditional checks), and letting the AI learn continuously from human corrections.
- **Understanding the Big Picture (Contextual Understanding):** The tools often miss the nuances of a project's business goals or unique rules. We need to build **context-aware models** that can read and understand project documents and history.
- **Security and Privacy Risks:** There are worries about proprietary code being exposed during AI training and the AI creating code with built-in security flaws. The solution involves developing privacy tools like **federated learning** and using security filters (like OWASP checks) on all AI suggestions.
- **Ethical and Legal Questions:** We still need to figure out who owns the **Intellectual Property (IP)** of AI-generated code. There are also concerns about job loss, especially for junior developers. Clear **ethical guidelines and policies** are essential.
- **Working Together (Human-AI Collaboration):** The goal isn't to replace developers, but to give them superpowers. Developers need to learn how to manage and control these AI systems effectively to ensure the code meets project goals, while also making sure they don't rely too heavily on the AI and lose fundamental coding skills.

3. METHODS

We've developed a smart, integrated AI system designed to make the software engineering process much faster and less error-prone. This framework is a **hybrid co-pilot** focusing on two key tasks: **writing code** and **intelligently fixing bugs**. It's built using advanced AI technology like CodeT5, GPT-5, and GraphCodeBERT.

3.1. Code Generation Module (CGM)

This is the creative part. It takes your simple request or high-level idea and instantly turns it into **functional code that understands its context**.

- **What it does:** It can autocomplete your lines, generate documentation comments (**docstrings**), and insert common code blocks (**boilerplate**).
- **How it works:** It produces several possible code options (**hypotheses**) and ranks them, checking which ones are most relevant, syntactically correct, and align with your past coding habits.

3.2. Debugging and Repair Module (DRM)

This is the quality control expert. It analyzes *all* your code—whether you wrote it or the CGM generated it—to pinpoint syntax and logic issues.

- **How it works:** Using advanced techniques like **Graph Neural Networks (GNNs)**, it identifies where a bug might

ISSN (Online): 2455-3662
EPRA International Journal of Multidisciplinary Research (IJMR) - Peer Reviewed Journal
Volume: 11| Issue: 11| November 2025|| Journal DOI: 10.36713/epra2013 || SJIF Impact Factor 2025: 8.691 || ISI Value: 1.188

be hiding, suggests multiple potential fixes, and cross-references those suggestions against a history of known vulnerabilities and patches.

Building Trust and Quality

To ensure developers can trust the suggestions and outputs, the framework includes essential safeguards:

- **Learning from You (Human-in-the-Loop):** Every suggestion comes with a **confidence score**. Developers are always in control and give continuous feedback (Accept, Modify, Reject). This feedback is used to automatically refine the AI models, making them better and more personalized over time.
- **Show Your Work (Explainable AI - XAI):** We don't want the AI to be a black box. The system provides a **reasoning** for its code suggestions by highlighting the most important parts of the code it looked at and showing you the AI's "thought process." This makes the system transparent.
- **Security Check:** Every piece of AI-generated code must pass through a strict **security filter**. A rule-based system checks the suggestions against established secure coding standards (like **OWASP**) to prevent the AI from introducing vulnerable code patterns.

Seamless Integration

This entire system is designed to fit right into your workflow. It works through simple plugins for popular environments like **Visual Studio Code** and **IntelliJ IDEA**, remembering your project details and session history to ensure the AI's outputs are always tailored to your specific project needs.

4. DISCUSSION

GAI tools, powered by **Large Language Models (LLMs)** like GitHub Copilot, demonstrate high efficacy in development tasks, significantly reducing the developer's **cognitive load** and speeding up the Software Development Lifecycle (SDLC).

- **Measurable Productivity:** Literature confirms GAI can reduce overall **development time and effort by up to 45%**.
- **Code Generation Quality:** Models show strong performance, with a top-1 accuracy reported at **83.2%** on standard benchmarks like HumanEval.
- **Quality Assurance:** AI tools (e.g., DeepCode, Amazon CodeGuru) enhance debugging and security, achieving a high bug localization accuracy of **91.4%** on datasets like Defects4J, and dramatically reducing average bug resolution time in real-time.
- **Maintainability:** GAI saves time by **automatically generating documentation and comments**, ensuring they remain current with the code.

These findings confirm GAI's immediate value in automating repetitive tasks and accelerating prototyping.

4.1 Technical Challenges and Trust Building

Despite compelling metrics, the reliability of GAI requires careful handling:

- **Accuracy and Context:** GAI can still produce **incorrect or suboptimal code**, making thorough human review and testing mandatory. A major limitation is the lack of **contextual understanding**—the AI struggles to grasp specific business logic, leading to technically sound but irrelevant suggestions. Future mitigation requires training models to ingest and understand project-specific documentation.
- **Explainable AI (XAI):** To build developer trust, **XAI is essential**. Providing a clear rationale for suggestions (e.g., highlighting influential code tokens) increases confidence in the AI's decisions, moving it away from being a black box.

4.2 Security, Ethics, and Over-Reliance

The GAI transformation is inseparable from significant ethical and security risks:

- **Security and Privacy:** Training on vast code corpora risks **exposing proprietary information** and may generate **insecure code patterns**. Organizations must adopt secure methods like **federated learning** and integrate **static analysis engines** to filter outputs against standards like **OWASP**.
- **Ethical Concerns:** Issues include potential **job displacement** for junior roles, the need for **bias mitigation**, and unresolved legal questions surrounding **Intellectual Property (IP)** ownership of AI-generated artifacts.
- **Skill Degeneration:** A critical risk is developer **over-reliance** on GAI, potentially leading to a degeneration of fundamental coding skills and critical thinking. GAI must be viewed as a supportive tool, not a replacement for core expertise.

4.3 Redefining Software Development Operations (SDOs)

The future of SDOs will be characterized by **hybrid models** moving from the Traditional (S1, human-owned roles) toward the ultimate goal of **Human-in-the-Loop (S4)**.

- In the S4 model, **AI manages development operations** (like testing and configurations), and human roles evolve to focus on **oversight, architectural design, quality assurance, and intervening** during novel issues.
- Achieving this requires developers to **upskill** to effectively manage and collaborate with sophisticated AI systems, viewing GAI as an **augmentation of human capability** that ensures outputs align with strategic goals and ethical guidelines.

4.4 GENERATIVE AI AND CODE QUALITY

Generative AI (GAI) is a game-changer for code quality in software development, essentially acting as an advanced quality assurance team. Tools like DeepCode and Amazon CodeGuru automatically scan codebases early on to catch bugs and security vulnerabilities before they ever reach production. Using sophisticated models like CodeBERT, the AI can often precisely locate a bug and even propose and apply fixes automatically, minimizing tedious human effort (with some frameworks showing a high bug localization accuracy of 91.4%). Furthermore, GAI accelerates the writing process itself: tools

ISSN (Online): 2455-3662
EPRA International Journal of Multidisciplinary Research (IJMR) - Peer Reviewed Journal
Volume: 11| Issue: 11| November 2025|| Journal DOI: 10.36713/epra2013 || SJIF Impact Factor 2025: 8.691 || ISI Value: 1.188

like GitHub Copilot quickly translate natural language into functional, syntactically correct code, ensuring consistency and adherence to standards across teams. GAI also actively helps maintain code quality by formatting code to project specifications and using tools like IntelliCode to suggest industry-based best practices. Finally, GAI significantly improves maintainability by performing automated code reviews and instantly generating accurate documentation and comments, making the code easier to understand and modify in the long run. Despite these huge benefits, developers must remain vigilant, as AI can occasionally suggest incorrect or insecure code, requiring careful human review and testing to ensure true quality and security.

4.5 IMPROVING OVERALL DEVELOPMENT EFFICIENCY

Generative AI (GAI) is a massive boost to **development efficiency**, streamlining the entire Software Development Lifecycle (SDLC) by taking over routine tasks and offering intelligent assistance.

4.6 How GAI Boosts Developer Productivity

GAI enhances efficiency through several key mechanisms:

- **Automation of Tedious Tasks:** GAI eliminates repetitive, time-consuming chores like writing boilerplate code, setting up basic project environments, or inserting standard functions. By handling these mundane activities, GAI frees developers to concentrate their effort on **complex, creative problem-solving**.
- **Accelerated Code Writing:** Tools like **GitHub Copilot** (using OpenAI Codex) and **TabNine** provide real-time code suggestions and autocompletion based on context. This significantly **reduces the cognitive load** and the time spent on manual code entry, directly accelerating development speed.
- **Rapid Prototyping:** GAI solutions compress the early-stage development process by quickly generating **functional prototypes** and design iterations based on simple, high-level specifications or natural language queries. This allows teams to quickly test ideas and discard suboptimal solutions.
- **Personalized Assistance:** Specialized AI tools learn an individual developer's **preferences, tendencies, and decisions**, allowing them to offer increasingly personalized and accurate suggestions for code snippets or refactoring. The longer the AI is used, the more effective and efficient it becomes.
- **Streamlined Quality Assurance:** GAI contributes to efficiency by automating code reviews and bug detection. This leads to **less time spent on manual debugging** and quality assurance, ultimately resulting in a shorter time-to-market.

4.7 Measurable Productivity Gains

Empirical data confirms the significant impact GAI has on reducing development effort:

- GAI can reduce the overall **development effort by up to 45%**.
- Users of tools like CopilotX report **autogenerating and testing 46%** of their code, accelerating their overall workflow by **55%**.
- AI-guided methods have been shown to drastically cut bug resolution time, dropping the average from 8.9 minutes per issue (manual) to **4.2 minutes**.
- The average code completion time has been observed to drop from 6.7 minutes with traditional methods to **3.1 minutes** with an integrated AI system.

5. CHALLENGES AND LIMITATIONS OF GENERATIVE AI

Generative AI (GAI) is transformative, but its integration into software development is met with several significant challenges and limitations that organizations must address to ensure responsible and reliable adoption.

5.1 Core Technical Imperfections

The primary limitations stem from the models' inherent capabilities and training data:

- Accuracy and Reliability: GAI doesn't always produce perfect code; it can generate incorrect or suboptimal code. Developers cannot blindly trust the outputs and must perform thorough human review and testing to prevent errors and maintain code quality.
- Contextual Understanding: GAI tools often struggle to grasp the broader project context, including complex business logic, specific requirements, or deep code dependencies. This can result in suggestions that are technically valid but contextually irrelevant or based on outdated patterns from the model's training data, requiring human modification.
- Error Propagation: While GAI reduces human coding mistakes, the AI itself can introduce its own errors. If developers fail to scrutinize the AI's buggy code, these errors can rapidly accumulate in the codebase, creating complicated problems that are difficult to resolve later. Security, Privacy, and Legal Risks: Integrating GAI raises critical concerns regarding sensitive information and compliance.
- Security of Generated Code: GAI models are trained on vast datasets that may contain insecure examples. This poses a risk of the AI generating code with inherent security flaws or vulnerable patterns that a programmer might miss.
- Proprietary Information Exposure: Relying on AI tools can risk inadvertently exposing sensitive or proprietary information during the model training or usage process, especially for critical systems like healthcare applications.
- Intellectual Property (IP) Dilemmas: There are persistent, unresolved legal and ethical issues concerning the ownership, copyright, and licensing of AI-generated code, particularly when models are trained on large, mixed-license code repositories.

ISSN (Online): 2455-3662
EPRA International Journal of Multidisciplinary Research (IJMR) - Peer Reviewed Journal
Volume: 11| Issue: 11| November 2025|| Journal DOI: 10.36713/epra2013 || SJIF Impact Factor 2025: 8.691 || ISI Value: 1.188

5.2 Ethical Concerns and Skill Impact

The widespread adoption of GAI has broader societal and professional implications:

- Over-reliance and Skill Degeneration: The ease of using AI suggestions poses a risk of developers adopting code without scrutiny, potentially leading to the degeneration of fundamental coding skills and critical thinking necessary for complex architectural design and advanced debugging.
- Job Displacement and Bias: Ethical dilemmas include the potential for job displacement (especially for junior developers) and the risk of the AI generating biased algorithms based on patterns in its training data, requiring rigorous efforts for fairness and compliance.

6. FUTURE PROSPECTS OF GENERATIVE AI

The future of Generative AI (GAI) in software development isn't just about faster coding; it's about making the AI more reliable, context-aware, and secure, fundamentally changing the developer's job.

Smarter Models and Deeper Context

To overcome current limitations, GAI models will get much smarter:

- Continuous Learning: Future models will be trained on larger and more diverse datasets and will constantly learn from developers' corrections and feedback. This is sometimes achieved through hybrid approaches, combining the AI's creativity with fixed, rule-based systems for stability.
- Context-Aware AI: AI won't just look at the code you're typing. It will evolve into specialized, domain-specific AI that can read and understand project-specific documentation, business requirements, and past project history. This means suggestions will finally be relevant to the *big picture*.
- Seamless Integration: GAI is expected to integrate fully into existing development processes, tying directly into DevOps practices, CI/CD cycles, and cloud platforms to enable self-optimizing software.

6.1 Security, Ethics, and New Roles

Addressing trust and ethical issues is paramount for mass adoption:

- Privacy-First Security: Future efforts will focus on privacy-preserving AI training techniques like federated learning to protect proprietary code. Organizations will establish robust auditing frameworks and industry-wide ethical guidelines to govern AI usage.
- The Evolved Developer: The developer's role will shift dramatically. Instead of spending time on repetitive coding tasks, humans will focus on high-level architectural design, critical thinking, quality assurance, and managing the AI system. The ultimate goal is a highly collaborative Human-AI environment where AI handles the routine, pushing development organizations toward highly automated operational scenarios.

7. CONCLUSION

Generative AI (GAI), driven by sophisticated Large Language Models (LLMs), has fundamentally transformed software engineering by seamlessly integrating into the coding process. Tools like GitHub Copilot, DeepCode, and Amazon CodeGuru now act as intelligent co-pilots, dramatically boosting productivity, code quality, and efficiency across the entire Software Development Lifecycle (SDLC).

7.1 The Big Benefits

Speed and Efficiency: GAI accelerates code generation, reduces the mental effort (cognitive load) on developers, and helps teams quickly build initial models (prototyping). This automation can lead to huge efficiency gains, with reports suggesting a potential reduction in development effort of up to 45%.

- Quality and Security: GAI improves code quality through automated code reviewing, security analysis, and enhanced maintainability by ensuring adherence to coding standards and generating accurate documentation.

7.2 Critical Challenges

Despite these gains, the technology introduces key hurdles that demand attention:

1. Trust Issues: GAI can generate incorrect or suboptimal code, meaning developers must maintain rigorous human review and testing.
2. Context Gap: AI often struggles to understand the complex business logic and specific requirements of a project, potentially leading to irrelevant suggestions.
3. Security and Privacy: There are serious concerns about exposing proprietary code during AI training and the risk of the AI generating insecure code patterns or known vulnerabilities.
4. Ethical and Legal Questions: Issues of Intellectual Property (IP) ownership, copyright for AI-generated artifacts, and the potential for job displacement must be addressed responsibly.

8. REFERENCES

1. ***Fan, Z., Wang, S., & Li, B. (2023)***. *Large language models for software engineering: Survey and open problems.*
2. ***Liu, C., Bunescu, R., & Storey, M. A. (2023).*** *Challenges and Opportunities of Generative AI for Software Engineering. IEEE Software*
3. ***Saravanan, V. et al. (2025)***. *Generative AI in Software Engineering: Revolutionizing Code Generation and Debugging.*
4. ***Bazzan, T. et al. (2024).*** *Analysing the Role of Generative AI in Software Engineering - Results from an MLR. DORAS | DCU Research Repository.*
5. ***Peng, B. et al. (2023)***. *A controlled experiment on the productivity of GitHub Copilot: Evidence from the JavaScript language.*
6. ***Sobania, A., et al. (2022).*** *An Empirical Study of GitHub Copilot's Code Suggestions. arXiv preprint arXiv:2202.04690.*

ISSN (Online): 2455-3662
EPRA International Journal of Multidisciplinary Research (IJMR) - Peer Reviewed Journal
Volume: 11| Issue: 11| November 2025|| Journal DOI: 10.36713/epra2013 || SJIF Impact Factor 2025: 8.691 || ISI Value: 1.188

7. ***Ziegler, J., et al. (2025).*** *How do Copilot Suggestions Impact Developers' Frustration and Productivity? arXiv preprint arXiv:2504.06808.*
8. ***Cavalcante, S., Ribeiro, E., & Oran, A. C. (2025).*** *The Impact of AI Tools on Software Development: A Case Study with GitHub Copilot and Other AI Assistants. SciTePress.*
9. ***Svyatkovskiy, H., Sundaresan, S., Fu, Y., & Sundaresan, N. (2020).*** *Intellicode Compose: Code Generation Using Transformer. arXiv preprint arXiv:2005.08025.*
10. ***Assessing the Quality and Security of AI-Generated Code:*** *A Quantitative Analysis. arXiv preprint arXiv:2508.14727 (August 2025).*
11. ***Ahmad, A., Chakraborty, A., & Mani, D. R. (2021).*** *A Transformer-Based Model for Fixing Bugs in Code. Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing (EMNLP).*
12. ***Wang, S. et al. (2022). D****etecting Vulnerabilities in Source Code Using CodeBERT and Graph Neural Networks. IEEE Transactions on Software Engineering, 48(6), 1785-1801.*
13. ***Zhu, P., Shi, Q., & Wang, D. (2023).*** *Secure Code Generation Using Adversarial Training. Proceedings of the 2023 IEEE Symposium on Security and Privacy (SP).*
14. ***Treude, C., & Storey, M. A. (2025).*** *Generative AI and Empirical Software Engineering: A Paradigm Shift. arXiv preprint arXiv:2502.08108.*
15. ***Schmidt, D. et al. (2024).*** *The Future of Software Engineering and Acquisition With Generative AI. Cutter Business Technology Journal, 37(2).*