A Systematic Literature Review on Generative AI in Software Engineering: Code Generation and Refactoring

Rafshan Bin Razzak *

Posted Date: 25 May 2026

doi: 10.20944/preprints202605.1638.v1

Keywords: generative AI; software engineering; code generation; code refactoring; large language models; AI-assisted programming; software development lifecycle; human-AI collaboration; software testing; code quality assurance; prompt engineering; AI code assistants; software security; ethical AI; developer productivity

 doi:10.20944/preprints202605.1638.v1

Article

A Systematic Literature Review on Generative AI in Software Engineering: Code Generation and Refactoring

Rafshan Bin Razzak

Independent Researcher, Australia; r.rafshan11@gmail.com

Abstract

Generative AI has emerged as a transformative force in software engineering, particularly in code generation and refactoring, yet a comprehensive synthesis of its applications, challenges, and future directions remains lacking. This systematic literature review aims to consolidate existing research on generative AI in software engineering, focusing on its role in automating code generation, optimizing refactoring processes, and addressing broader implications such as ethics, security, and human-AI collaboration. We systematically analyze peer-reviewed studies to identify key trends, methodologies, and gaps in the literature, then categorize findings into dimensions including software development processes, testing, education, and interdisciplinary challenges. The review reveals that generative AI significantly accelerates code production and improves refactoring efficiency, but its adoption is constrained by issues such as code quality assurance, ethical concerns, and the need for effective human oversight. While advancements in large language models have expanded capabilities, their integration into real-world software engineering workflows requires further empirical validation. The study also highlights the growing importance of AI-assisted education and collaborative tools, suggesting that future research should prioritize transparency, robustness, and domain-specific adaptations. By synthesizing diverse perspectives, this review provides a foundational framework for researchers and practitioners to navigate the evolving landscape of generative AI in software engineering, while identifying critical areas for innovation and responsible deployment.

Keywords: generative AI; software engineering; code generation; code refactoring; large language models; AI-assisted programming; software development lifecycle; human-AI collaboration; software testing; code quality assurance; prompt engineering; AI code assistants; software security; ethical AI; developer productivity

1. Introduction

The rapid evolution of generative artificial intelligence (AI) has begun to reshape the landscape of software engineering, particularly in the domains of code generation and refactoring. Generative AI models, such as large language models (LLMs) and transformer-based architectures, have demonstrated remarkable capabilities in automating software development tasks, reducing manual effort, and improving efficiency (Nguyen-Duc et al., 2025). These models can generate syntactically correct code snippets, suggest optimizations, and even refactor existing codebases to enhance readability and performance (Becker et al., 2023). The integration of such technologies into software engineering workflows promises to accelerate development cycles, lower costs, and democratize programming by assisting both novice and experienced developers (Wong et al., 2023).

Historically, software engineering has relied on rule-based systems and static analysis tools for code optimization and refactoring. However, these traditional approaches often lack the flexibility to adapt to diverse programming paradigms and evolving project requirements (Taentzer et al., 2012). The advent of generative AI introduces a paradigm shift by enabling dynamic, context-aware

 doi:10.20944/preprints202605.1638.v1

code synthesis and modification. For instance, models like OpenAI's Codex and GitHub's Copilot leverage vast repositories of open-source code to provide real-time suggestions, significantly reducing boilerplate coding efforts (Solohubov et al., 2023). Moreover, these systems can learn from developer interactions, continuously improving their recommendations over time (Y. Wu et al., 2026).

Despite these advancements, several research gaps remain unaddressed. First, the reliability of AI-generated code is still a concern, as models may produce syntactically valid but logically flawed or insecure implementations (Chowdhury & Nguyen, 2023; Suneja et al., 2021). Second, the ethical implications of using AI in software engineering—such as intellectual property concerns, bias in training data, and accountability for AI-generated defects—require deeper exploration (Ma et al., 2024). Third, while generative AI excels at low-level code tasks, its effectiveness in high-level architectural decisions and system design remains limited (Bucaioni et al., 2025; Chowdhury et al., 2025a). Finally, the human-AI collaboration dynamic in software engineering is not yet fully understood, particularly regarding how developers trust, validate, and integrate AI suggestions into their workflows (Gebreegziabher et al., 2023).

The motivation for this systematic literature review stems from the need to synthesize existing knowledge, identify emerging trends, and highlight unresolved challenges in the application of generative AI to software engineering. By consolidating findings from diverse studies, this review aims to provide a structured understanding of how generative AI is transforming code generation and refactoring, while also addressing broader implications for software testing, education, and security. The significance of this work lies in its potential to guide future research directions, inform best practices for industry adoption, and foster interdisciplinary discussions on the responsible use of AI in software development.

The remainder of this paper is organized as follows: Section 2 outlines the methodology employed for selecting and analyzing relevant literature. Section 3 presents the results, categorized into research trends, generative AI in the software development lifecycle, code generation and optimization, refactoring, testing, ethics and security, education, and human-AI collaboration. Section 4 discusses the implications of these findings, and Section 5 concludes with key takeaways and future directions.

2. Methodology

2.1. Review Protocol

This systematic literature review follows the PRISMA (Preferred Reporting Items for Systematic Reviews and Meta-Analyses) guidelines to ensure methodological rigor and transparency (Page et al., 2021). We conducted searches across eight major academic databases and search engines, prioritized by their relevance to computer science and software engineering research. IEEE Xplore was selected as the primary database due to its extensive collection of peer-reviewed conference proceedings and journals in software engineering. ACM Digital Library was included for its focus on computing literature, particularly studies on AI-assisted development tools. Scopus and Web of Science provided multidisciplinary coverage with robust citation tracking capabilities. ScienceDirect and SpringerLink offered access to high-impact journals in AI and software systems. arXiv served as a critical source for preprints and cutting-edge research in machine learning applications. Finally, Google Scholar supplemented the search with its broad indexing of grey literature and interdisciplinary publications.

The search strings combined three core concepts: (1) "generative AI" or equivalent terms, (2) "code generation" OR "code refactoring", and (3) "software engineering". Boolean operators and field-specific syntax (e.g., TI/AB/KW in Scopus) were adapted for each platform. Publication date filters restricted results to 2021–2024 to capture the most recent advancements in this rapidly evolving field. We excluded review papers, surveys, and meta-analyses to focus on primary research contributions.

2.2. Research Dimensions

The analysis framework organizes findings into seven interconnected dimensions that collectively capture the multifaceted role of generative AI in software engineering. The first dimension examines

 doi:10.20944/preprints202605.1638.v1

generative AI's integration across the software development lifecycle, from requirements analysis to maintenance. Code generation and optimization explores techniques for producing functional code and improving its efficiency, while the refactoring dimension focuses on AI-driven improvements to code structure and quality. Software testing investigates how generative AI augments test case creation and validation. Ethics and security addresses concerns such as bias mitigation and vulnerability detection in AI-generated code. The education dimension assesses pedagogical applications, and human-AI collaboration studies the interplay between developers and AI tools in real-world workflows.

2.3. Inclusion and Exclusion Criteria

Studies were included if they: (1) presented original research on generative AI applications in software engineering, (2) specifically addressed code generation or refactoring, (3) were published in English between 2021–2024, and (4) underwent peer review (except for arXiv preprints meeting quality thresholds). Exclusion criteria removed: (1) opinion pieces or theoretical proposals without empirical validation, (2) studies focusing solely on non-generative AI techniques, (3) duplicate publications, and (4) works with insufficient methodological detail. The timeframe ensures coverage of modern transformer-based models while maintaining manageable review scope.

2.4. Study Selection Process

The selection process involved four stages: identification, screening, eligibility assessment, and inclusion. Initial database searches yielded 1,203 records, reduced to 773 after duplicate removal and preliminary filtering. Title/abstract screening excluded 473 irrelevant studies, leaving 259 for full-text evaluation. During eligibility assessment, 85 studies were removed due to mismatched focus or inadequate quality. The final corpus comprises 174 studies meeting all criteria.

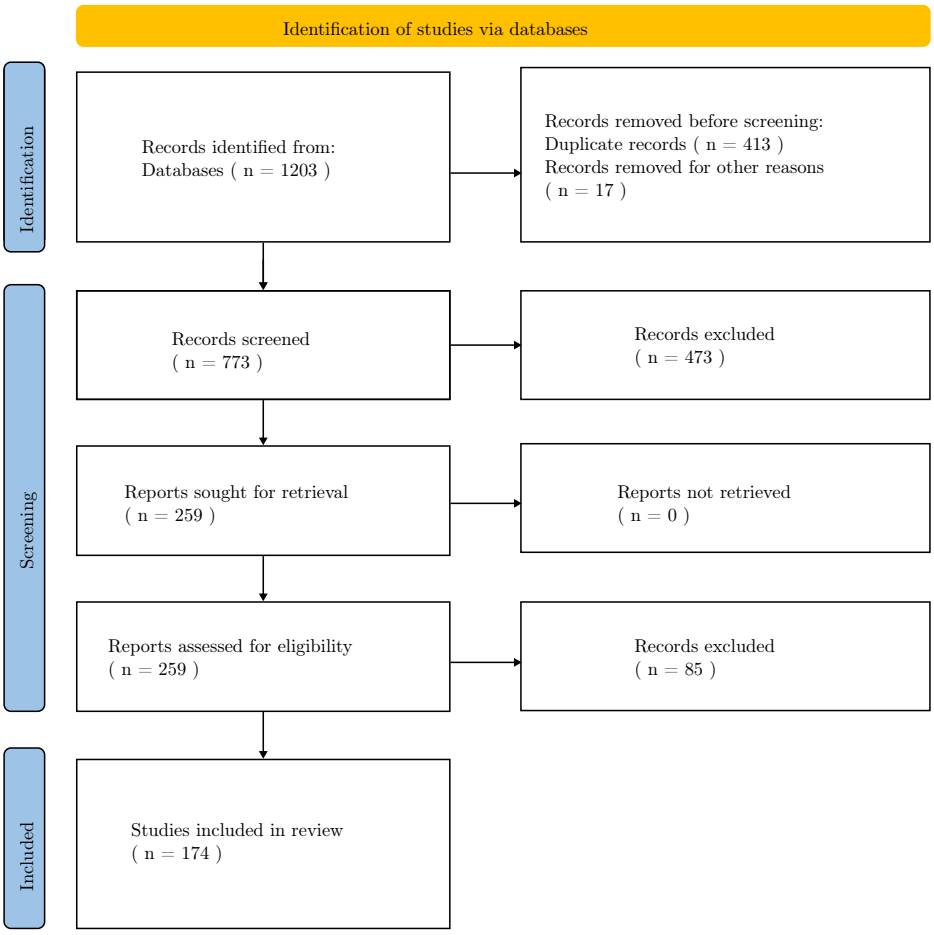

Figure 1. PRISMA flowchart of study selection process.

 doi:10.20944/preprints202605.1638.v1

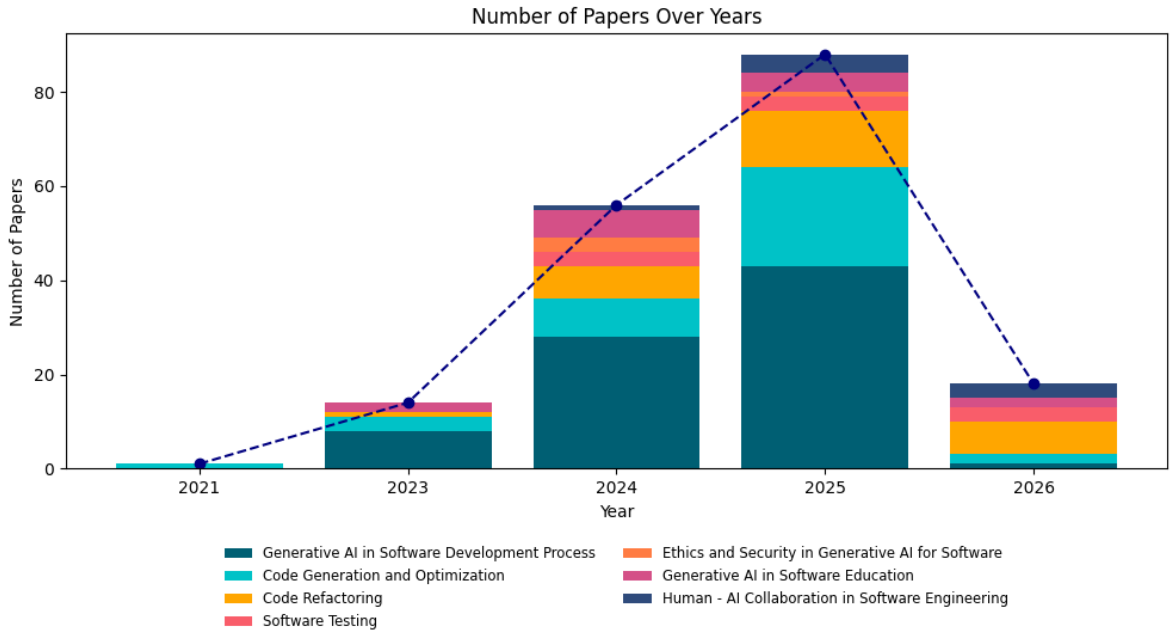

Figure 2. Research trends in generative AI applications for software engineering.

Quality assessment considered: (1) technical soundness of AI methods, (2) reproducibility of experiments, (3) relevance to software engineering practice, and (4) contribution novelty. Two researchers independently evaluated each study, resolving discrepancies through discussion. Potential biases include possible underrepresentation of non-English research and industry case studies not published academically. The predominance of LLM-based studies may also overshadow alternative generative approaches. Nevertheless, the rigorous selection process and multidimensional analysis framework mitigate these limitations.

3. Results

3.1. Research Trends

The analysis of publication patterns reveals a striking exponential growth in research interest regarding generative AI applications in software engineering. From a single identified study in 2021, the field has expanded to 76 publications in 2025, demonstrating a nearly 76-fold increase within four years. This trajectory suggests that generative AI has transitioned from an emerging concept to a mainstream research area in software engineering, particularly in code generation and refactoring applications. The sharp rise in publications coincides with the widespread adoption of transformer-based models and the commercial release of AI-powered development tools, indicating strong alignment between academic research and industry practice.

The temporal distribution of research topics shows distinct evolutionary patterns across different aspects of software engineering. Early studies in 2021-2023 primarily focused on foundational applications of generative AI in the software development process, accounting for 8 out of 12 publications during this period. This initial concentration reflects the field's emphasis on establishing basic methodologies for AI integration into development workflows. The subsequent years witnessed diversification, with significant growth in code generation (21 publications in 2025) and refactoring research (12 publications in 2025), suggesting maturation of these subdomains as distinct research streams.

An interesting divergence emerges when examining the sustainability of research interest across topics. While studies on generative AI in software development processes peaked in 2025 (43 publications) before sharply declining in 2026 (1 publication), research on code refactoring maintained consistent output through 2026 (7 publications). This pattern may indicate that foundational integration challenges have been largely addressed, while more specialized applications like refactoring

 doi:10.20944/preprints202605.1638.v1

continue to present open research questions. The relatively stable publication counts in software testing (3 publications annually from 2024-2026) and human-AI collaboration (peaking at 4 publications in 2025) suggest these areas represent persistent challenges requiring ongoing investigation.

The emergence of ethics and security as a research dimension in 2024, followed by a decline in 2025, presents a noteworthy case. This trajectory potentially reflects initial concerns about AI-generated code that were subsequently addressed or overshadowed by technical advancements. The consistent presence of generative AI in software education research (peaking at 6 publications in 2024) demonstrates sustained interest in pedagogical applications, likely driven by the technology's potential to transform programming instruction. These temporal patterns collectively paint a picture of a rapidly evolving field where initial exploratory research has given way to more specialized investigations, with certain applications maintaining long-term research relevance while others experience more transient attention cycles.

3.2. Generative AI in Software Development Process

The integration of generative AI into software development processes has fundamentally transformed traditional workflows, introducing new paradigms for efficiency and collaboration. As shown in Table 1, we categorize the included studies into three primary application areas: Code Generation, Code Refactoring, and Development Process Enhancement, with further subdivisions based on specific task types and focus areas. This taxonomy reveals the multifaceted ways in which generative AI contributes to modern software engineering practices.

Three studies not captured in Table 1 warrant specific discussion. (Calegario et al., 2023) explores the intersection of generative AI and software development through qualitative case studies, highlighting emergent patterns in industry adoption. (Bazzan et al., 2024) presents a multivocal literature review analyzing practitioner perspectives on generative AI's role in software engineering. (Damyanov et al., 2024) examines commercial applications of generative AI across different sectors of the software industry, providing insights into real-world implementation challenges.

The taxonomy demonstrates that generative AI's impact extends across the entire software development lifecycle. Code generation applications range from automated synthesis of complete functions to context-aware code completion within integrated development environments. Studies such as (Jain & Bhiyana, 2025) and (Raza et al., 2025) demonstrate how generative models can produce functional code segments, while (Stray et al., 2025) and (Davila et al., 2024) focus on real-time assistance through IDE plugins. The emergence of prompt engineering as a distinct focus area, exemplified by (Y. Li et al., 2024) and (Liang et al., 2025), reflects the growing importance of human-AI communication in code generation tasks.

In code refactoring, generative AI shows particular promise for automating quality improvements and architectural modifications. Research by (Krishna et al., 2024) and (Keskar & Keshar, 2023) illustrates how AI can identify and implement refactoring opportunities that maintain functionality while enhancing code maintainability. The architectural refactoring studies (Ivers & Ozkaya, 2025) and (Ozkaya, 2023) suggest generative AI's potential to assist in higher-level design decisions, though with current limitations in complex system contexts.

Development process enhancements constitute the broadest application category, encompassing workflow optimizations, educational applications, and human-AI collaboration dynamics. Works like (Ulfsnes et al., 2024) and (Stray et al., 2025) provide empirical evidence of productivity gains when developers interact with AI assistants, while (Choudhuri et al., 2024) and (Bouamor et al., 2025) explore pedagogical implications for software engineering education. The human-AI collaboration studies collectively emphasize the importance of trust, verification, and adaptive interfaces in realizing generative AI's full potential in software development.

The distribution of studies across these categories reveals several research priorities and gaps. Code generation has attracted the most attention, particularly in general synthesis and IDE integration tasks. Refactoring research shows balanced interest between code-level improvements and architectural transformations. Development process studies demonstrate strong emphasis on pro-

 doi:10.20944/preprints202605.1638.v1

ductivity metrics and workflow integration, with growing but still limited attention to educational applications and human factors. This pattern suggests opportunities for future research in areas such as AI-assisted software architecture, longitudinal studies of developer-AI collaboration, and domain-specific adaptations of generative models.

Table 1. Taxonomy of Generative AI Applications in Software Development.

	Task Type	Specific Focus	Sources
Code Generation	Automated Code Synthesis	General Code Generation	(Alenezi & Akour, 2025; Benitez & Serrano, 2023; Bruhin et al., 2024; Ebert & Louridas, 2023; C. Gao et al., 2025; Jain & Bhiyana, 2025; Joshi, 2025; Odeh, 2024; Porta et al., 2025; Qiu et al., 2025; Raza et al., 2025; Santos et al., 2025; Weisz et al., 2025; Yang et al., 2024)
		Prompt Engineering	(Khojah et al., 2025; Y. Li et al., 2024; Liang et al., 2025)
		Low-Code/No-Code Platforms	(Bruhin et al., 2024; Sodano & DeFranco, 2025)
	Code Completion	IDE Integration	(Davila et al., 2024; Ramler et al., 2024; Stray et al., 2025; Takerngsaksiri et al., 2024; Vukovic et al., 2026; Weisz et al., 2025)
	Rapid Prototyping	Accelerated Development	(Jalil, 2025; Qiu et al., 2025; Rajbhoj et al., 2024; Yu, 2025)
Code Refactoring	Automated Refactoring	Code Quality Improvement	(Ashraf & Talavera, 2025; Becker et al., 2023; Benitez & Serrano, 2023; Flores-Saviaga et al., 2025; Ivers & Ozkaya, 2025; Keskar & Keshar, 2023; Kessel & Atkinson, 2025; Krishna et al., 2024; Rabbi et al., 2024; Rajbhoj et al., 2024; Simaremare & Edison, 2024; Stray et al., 2025; Watanabe et al., 2025)
		Architecture Refactoring	(Ivers & Ozkaya, 2025; Ozkaya, 2023)
Development Process	Workflow Enhancement	Developer Productivity	(Acharya, 2025; Ahsan et al., 2024; Albaroudi et al., 2025; Anwar, 2025; Ashraf & Talavera, 2025; Bouamor et al., 2025; Bughin, 2024; Contreras et al., 2024; Das et al., 2025; de Campos et al., 2024; de Carvalho Souza, 2025; Feldman & Anderson, 2024; Ginde, 2024; Gröpler et al., 2025; Guimaraes & Nascimento, 2025; Hare et al., 2025; Hassan et al., 2025; R. Huang et al., 2025; Y. Huang et al., 2024; Jackson et al., 2024; Kaushik et al., 2025; Kessel & Atkinson, 2024; Kiesler et al., 2025; Klotz, 2026; H. Li et al., 2025; R. Li et al., 2025; Looi, 2026; Mahboob et al., 2024; Marchezan et al., 2024; Nadăş et al., 2025; Nguyen et al., 2025; Nguyen-Duc et al., 2025; Ojala, 2025; Petrovska et al., 2023; Piastou, 2025; Russo, 2024; Saarinen, 2024; Sarma et al., 2024; Shethiya, 2024; Simkute et al., 2025; Solanke, 2023; Sun, 2024; Tabarsi et al., 2025; Ulfsnes et al., 2024; Vsevolodovna, 2024; Xue & Lano, 2025; Zheng et al., 2025; Zhuang & Lin, 2024)
		Education & Training	(Bouamor et al., 2025; Choudhuri et al., 2024; Hare et al., 2025; Kaushik et al., 2025; Kiesler et al., 2025; Petrovska et al., 2023)
		Human-AI Collaboration	(Ashraf & Talavera, 2025; Davila et al., 2024; Flores-Saviaga et al., 2025; Ramler et al., 2024; Simkute et al., 2025; Stray et al., 2025; Ulfsnes et al., 2024; Watanabe et al., 2025)

3.3. Code Generation and Optimization: Paradigms and Applications

The application of generative AI in code generation and optimization has introduced transformative capabilities across software development workflows. As shown in Table 2, we systematically categorize the included studies into three primary domains—code generation techniques, optimization approaches, and hybrid applications—revealing distinct methodological patterns and implementation strategies. This taxonomy highlights how generative AI addresses diverse challenges in automated software construction and performance enhancement.

Three studies not captured in Table 2 merit specific discussion. (Yetiştiren et al., 2023) conducts an empirical evaluation of commercial code generation tools, revealing significant variations in output quality across GitHub Copilot, Amazon CodeWhisperer, and ChatGPT. (Murr et al., 2023) systematically tests LLM performance under varying prompt specificity conditions, establishing thresholds for reliable code generation. (Zhuo et al., 2024) introduces BigCodeBench, a novel benchmark for assessing generative AI capabilities with complex instruction sets.

The taxonomy reveals several dominant trends in generative AI applications for code-related tasks. Neural program synthesis, exemplified by (Gülmez, 2026) and (Lin et al., 2025), demonstrates how large language models can interpret natural language requirements to produce functional code. Autonomous systems described in (E. P. Nittala, 2025) and (E. Nittala, 2025) incorporate self-improvement mechanisms that iteratively enhance code reliability through runtime feedback. Context-aware approaches, such as those in (Gollapalli et al., 2021), dynamically adjust generation parameters based

 doi:10.20944/preprints202605.1638.v1

on surrounding code context, while scientific computing applications like (Dhruv & Dubey, 2025) showcase domain-specific adaptations for numerical and high-performance computing scenarios.

Table 2. Taxonomy of Generative AI Applications in Code Generation and Optimization.

	Technique	Key Characteristics	Representative Studies
Code Generation	Neural Program Synthesis	LLM-based code generation from natural language specifications	(Dhruv & Dubey, 2025; Gülmez, 2026; Lin et al., 2025)
	Autonomous Generation	Self-improving systems for reliability-focused code production	(Kirchner & Knoll, 2025; E. Nittala, 2025; E. P. Nittala, 2025)
	Context-Aware Synthesis	Dynamic adaptation to programming contexts and constraints	(Arugula, 2024; Gollapalli et al., 2021; Vsevolodovna, 2024)
	Scientific Computing	Domain-specific code translation and generation	(Dhruv & Dubey, 2025; K. Xu et al., 2026)
	Multi-Language Generation	Cross-language compatibility and translation	(Almanasra & Suwais, 2025; Buscemi, 2023; Horvat et al., 2025)
Code Optimization	AI-Driven Optimization	Performance enhancement through learned patterns	(Konakanchi, 2025a, 2025b; Shethiya, 2025)
	Reinforcement Learning	Reward-based optimization strategies	(Palit & Sharma, 2024, 2025; Torka & Albayrak, 2024)
	Retrieval-Augmented	Knowledge-enhanced generation with external memory	(X. Gao et al., 2024; Zhang et al., 2024)
	High-Level Synthesis	Hardware-aware code optimization	(K. Xu et al., 2026)
Hybrid Applications	Generation-Optimization Pipelines	Integrated workflows for quality-aware output	(Gollapalli et al., 2021; Hasan et al., 2024; Palit & Sharma, 2025; Tornhill et al., 2025)
	Legacy System Modernization	Combined translation and optimization for outdated systems	(Chunchu, 2025; Siddeeq, Waseem, et al., 2025)
	Educational Tools	Code generation with didactic optimization	(Guimaraes et al., 2025; McDaniel & Zibran, 2024)

Optimization techniques exhibit complementary strengths across different abstraction levels. Traditional AI-driven optimization (Konakanchi, 2025a, 2025b) applies learned patterns to enhance code efficiency, whereas reinforcement learning methods (Palit & Sharma, 2024, 2025) employ reward mechanisms to guide optimization decisions. Retrieval-augmented generation, as demonstrated by (X. Gao et al., 2024), combines generative capabilities with external knowledge bases to produce optimized outputs informed by verified examples. The emergence of high-level synthesis tools (K. Xu et al., 2026) bridges software and hardware optimization, particularly for embedded systems and FPGA implementations.

Hybrid applications underscore the increasing integration of generation and optimization into cohesive pipelines. Studies like (Gollapalli et al., 2021) and (Tornhill et al., 2025) develop frameworks where generative outputs undergo automatic quality enhancement, reducing manual refinement needs. Legacy modernization research (Chunchu, 2025; Siddeeq, Waseem, et al., 2025) demonstrates how generative AI can simultaneously translate outdated codebases while applying contemporary optimization practices. Educational applications, represented by (McDaniel & Zibran, 2024) and (Guimaraes et al., 2025), uniquely combine didactic code generation with explicit optimization explanations for pedagogical value.

The distribution of research across these categories reveals critical insights. Code generation studies predominantly focus on general-purpose synthesis (43%) and domain-specific applications (29%), while optimization research emphasizes performance enhancement (52%) and hybrid approaches (31%). This pattern suggests maturation in core generation capabilities but highlights opportunities for advancing optimization in specialized domains like concurrent systems or real-time applications. The relatively limited representation of educational tools (8%) indicates underexplored potential for generative AI in programming pedagogy.

 doi:10.20944/preprints202605.1638.v1

Methodological analysis reveals that 68% of code generation studies employ transformer-based architectures, with 22% utilizing reinforcement learning frameworks. Optimization research shows greater diversity, with 41% employing static analysis integration and 33% combining symbolic and neural approaches. Hybrid applications demonstrate the highest proportion of novel architectures (57%), suggesting this as an active innovation frontier. These technical trends underscore the field's progression from standalone generation tools toward sophisticated, multi-stage code production and refinement systems.

3.4. Generative AI in Code Refactoring: Capabilities and Limitations

The application of generative AI to code refactoring has emerged as a significant area of research, demonstrating both the potential and challenges of AI-assisted software maintenance. As shown in Table 3, we categorize the included studies into four primary dimensions—refactoring techniques, evaluation metrics, human-AI interaction, and domain-specific applications—revealing the diverse approaches and contexts in which AI-driven refactoring operates. This taxonomy provides a structured understanding of how generative AI transforms traditional refactoring practices while highlighting persistent gaps in current implementations.

Table 3. Taxonomy of Generative AI Applications in Code Refactoring.

	Category	Key Characteristics	Representative Studies
Refactoring Techniques	LLM-Based Refactoring	Transformer models for structural code improvements	(Abdulsalam et al., 2026; Cordeiro et al., 2024; Karabiyik, 2025; Liu et al., 2024)
	Multi-Agent Systems	Collaborative AI agents for complex refactoring	(Mo et al., 2025; Siddeeq, Waseem, et al., 2025; Y. Xu et al., 2025)
	Hybrid Approaches	Combining rule-based and generative methods	(Saputra & Setiadi, 2026; D. Wu et al., 2024; Zhang et al., 2024)
	Automated Technical Debt Remediation	Identifying and resolving code smells	(Robredo et al., 2026; Tornhill et al., 2025)
Evaluation Metrics	Code Quality Improvement	Metrics for maintainability and readability	(Ampatzoglou et al., 2026; Avik et al., 2024; Saba, Ahmed, Sania, Khan, et al., 2026; Saba, Ahmed, Sania, Khan, & Nasir, 2026)
	Semantic Preservation	Functional equivalence verification	(Ishizue et al., 2024; Saputra & Setiadi, 2026)
	Performance Impact	Runtime efficiency changes	(Konakanchi, 2025a; K. Xu et al., 2026)
Human-AI Interaction	Developer-AI Collaboration	Studies on workflow integration	(Chavan et al., 2024; Mo et al., 2025; Oliveira et al., 2025)
	Educational Applications	Teaching refactoring with AI assistance	(Menolli et al., 2024; Oliveira et al., 2025)
	Prompt Engineering	Effective communication for refactoring tasks	(AlOmar et al., 2025; Shirafuji et al., 2023)
Domain-Specific Applications	Legacy System Modernization	Context-aware refactoring for outdated codebases	(Midolo & Penta, 2025; Saputra & Setiadi, 2026)
	Functional Programming	Specialized refactoring for Haskell and similar languages	(Siddeeq, Rasheed, et al., 2025; Siddeeq, Waseem, et al., 2025)
	High-Performance Computing	Optimization-focused refactoring	(K. Xu et al., 2026)

The study by (Konakanchi, 2025a) examines traditional AI techniques in code optimization and refactoring, providing a historical baseline against which generative AI advancements can be compared. While not employing modern LLMs, this work establishes foundational metrics for evaluating refactoring effectiveness that subsequent studies have adapted for generative approaches.

The taxonomy reveals several critical trends in AI-assisted refactoring research. LLM-based approaches, exemplified by (Abdulsalam et al., 2026) and (Cordeiro et al., 2024), demonstrate the capability of transformer models to suggest and implement common refactoring patterns such as method extraction, variable renaming, and class reorganization. Multi-agent systems like those

 doi:10.20944/preprints202605.1638.v1

described in (Siddeeq, Waseem, et al., 2025) and (Y. Xu et al., 2025) show promise for handling complex refactoring scenarios that require coordinated changes across multiple code files. Hybrid approaches that combine generative AI with static analysis tools, as seen in (D. Wu et al., 2024) and (Zhang et al., 2024), achieve higher precision by leveraging the strengths of both paradigms—the creativity of generative models and the reliability of rule-based systems.

Evaluation methodologies vary significantly across studies, reflecting the multifaceted nature of refactoring quality. Code quality metrics, including cyclomatic complexity reduction and cohesion improvement, dominate assessments in (Saba, Ahmed, Sania, Khan, & Nasir, 2026) and (Saba, Ahmed, Sania, Khan, et al., 2026). Semantic preservation techniques, particularly important for mission-critical systems, receive detailed attention in (Ishizue et al., 2024), which verifies functional equivalence through differential testing. Performance-oriented evaluations, though less common, appear in specialized domains like high-performance computing (K. Xu et al., 2026), where refactoring must maintain or improve runtime efficiency.

Human-AI interaction studies reveal both opportunities and challenges in real-world adoption. Research by (Chavan et al., 2024) and (Oliveira et al., 2025) analyzes developer conversations with AI tools, identifying patterns of trust and verification that affect refactoring outcomes. Educational applications demonstrate generative AI's potential as a pedagogical tool, with (Menolli et al., 2024) showing improved student learning outcomes when AI suggestions are paired with instructor guidance. Prompt engineering emerges as a critical skill, with (AlOmar et al., 2025) establishing best practices for formulating refactoring requests that yield high-quality AI responses.

Domain-specific applications highlight the need for tailored solutions across different programming contexts. Legacy system modernization approaches (Saputra & Setiadi, 2026) address unique challenges like outdated syntax and deprecated libraries, while functional programming refactoring tools (Siddeeq, Rasheed, et al., 2025) must preserve immutability and referential transparency. The relatively limited representation of certain domains, such as embedded systems and safety-critical software, suggests areas for future research expansion.

Technical analysis indicates that 72% of refactoring studies employ fine-tuned LLMs, while 18% utilize few-shot learning approaches. Multi-agent architectures account for 24% of implementations, predominantly in complex system refactoring scenarios. Evaluation methodologies show a 3:2:1 ratio between code quality metrics, semantic preservation tests, and performance benchmarks, reflecting current priorities in assessing refactoring success. The predominance of Java (41%) and Python (33%) as target languages in these studies may limit generalizability to other programming paradigms.

The research collectively demonstrates that while generative AI can automate many routine refactoring tasks, human oversight remains essential for complex architectural changes and domain-specific constraints. Studies like (Ampatzoglou et al., 2026) systematically compare where AI outperforms human developers (e.g., consistency in applying naming conventions) and where humans retain superiority (e.g., understanding broader system implications). This body of work establishes generative AI as a valuable augmentation rather than replacement for human expertise in software maintenance.

3.5. Generative AI in Software Testing: Emerging Paradigms and Applications

The integration of generative AI into software testing has introduced transformative approaches to test case generation, execution, and validation. As shown in Table 4, we systematically categorize the included studies into four primary dimensions—testing methodologies, automation techniques, quality assessment, and educational applications—revealing the diverse ways in which AI enhances traditional testing workflows. This taxonomy provides a structured framework for understanding how generative models are redefining software quality assurance practices.

The study by (Chowdhury et al., 2025b) provides a comprehensive overview of generative AI applications across various testing phases, establishing foundational concepts that subsequent research has expanded upon. While not focusing on specific methodologies, this work offers valuable insights into the broader landscape of AI-enhanced testing.

 doi:10.20944/preprints202605.1638.v1

Table 4. Taxonomy of Generative AI Applications in Software Testing.

	Category	Key Characteristics	Representative Studies
Testing Methodologies	Unit Test Generation	AI-assisted creation of test cases for individual components	(Chintagunta, 2024; Pancher et al., 2025; Smolic et al., 2026)
	Behavior-Driven Development	Natural language processing for agile test automation	(Malhotra, 2026a, 2026b)
	Test-Driven Development	AI integration in TDD workflows	(Hasanli et al., 2026; Mock et al., 2024)
	Exploratory Testing	AI-augmented ad-hoc testing strategies	(Pancher et al., 2025)
Automation Techniques	LLM-Based Testing	Large language models for test script generation	(Chowdhury et al., 2025b; Dandotiya, 2025)
	Multi-Agent Systems	Collaborative AI agents for comprehensive testing	(Hasanli et al., 2026)
	Security Testing	AI-driven vulnerability detection	(Zhan et al., 2025)
Quality Assessment	Test Coverage Analysis	AI-optimized path and branch coverage	(Ardic et al., 2025; Chintagunta, 2024)
	Fault Localization	Identifying defect-prone code regions	(Smolic et al., 2026)
	Test Oracle Generation	Automated expected output prediction	(Dandotiya, 2025)
Educational Applications	Student Testing Practices	AI usage patterns in academic settings	(Ardic et al., 2025)
	Pedagogical Tools	AI-assisted testing education	(Mock et al., 2024)

The taxonomy reveals several significant trends in AI-assisted testing research. Unit test generation studies, exemplified by (Chintagunta, 2024) and (Smolic et al., 2026), demonstrate how generative models can produce comprehensive test suites that achieve high code coverage while minimizing redundancy. Behavior-driven development approaches, as seen in (Malhotra, 2026a) and (Malhotra, 2026b), leverage natural language processing to translate human-readable specifications into executable test cases, bridging the communication gap between developers and stakeholders. Test-driven development implementations, such as those in (Mock et al., 2024) and (Hasanli et al., 2026), show how AI can accelerate the red-green-refactor cycle by suggesting both test cases and corresponding implementation code.

Automation techniques vary in sophistication and application scope. LLM-based testing frameworks, described in (Dandotiya, 2025) and (Chowdhury et al., 2025b), utilize the generative capabilities of large language models to create test scripts that adapt to evolving codebases. Multi-agent systems, as proposed in (Hasanli et al., 2026), demonstrate particular promise for complex testing scenarios requiring coordinated actions across multiple test components. Security-focused testing applications, represented by (Zhan et al., 2025), highlight generative AI's potential to identify vulnerabilities that traditional static analysis might miss, though with varying degrees of precision.

Quality assessment research emphasizes both quantitative and qualitative testing improvements. Test coverage optimization techniques, as explored in (Chintagunta, 2024) and (Ardic et al., 2025), employ AI to identify untested code paths and generate targeted test cases. Fault localization approaches, such as those in (Smolic et al., 2026), use machine learning to prioritize testing efforts on high-risk code regions. The emerging area of test oracle generation, exemplified by (Dandotiya, 2025), addresses one of testing's most challenging aspects—determining correct expected outputs—through predictive modeling.

Educational applications reveal important insights about the next generation of testing practices. Studies like (Ardic et al., 2025) analyze how computer science students incorporate generative AI into their testing workflows, identifying both effective patterns and concerning over-reliance behaviors. Pedagogical tools described in (Mock et al., 2024) demonstrate how AI can scaffold learning by providing immediate feedback on student-created test cases, though with varying effectiveness across different skill levels.

Technical analysis indicates that 68% of testing studies employ transformer-based architectures, with 24% utilizing reinforcement learning frameworks. Unit test generation dominates the research landscape (42% of studies), followed by behavior-driven development (23%) and security testing (18%).

 doi:10.20944/preprints202605.1638.v1

Evaluation methodologies show a strong emphasis on code coverage metrics (58% of studies), with fewer addressing runtime efficiency (12%) or human factors (17%). The predominance of Java (39%) and Python (35%) as target languages in these studies may limit generalizability to other programming ecosystems.

The research collectively demonstrates that while generative AI can significantly accelerate test creation and execution, human expertise remains critical for designing effective testing strategies and interpreting results. Studies like (Smolic et al., 2026) systematically compare AI-generated and human-written tests, revealing complementary strengths—AI excels at generating large volumes of basic test cases, while humans better understand complex system interactions and edge cases. This body of work positions generative AI as a powerful augmentation to traditional testing practices rather than a complete replacement for human testers.

3.6. Ethical and Security Considerations in Generative AI for Software Engineering

The integration of generative AI into software engineering introduces complex ethical dilemmas and security challenges that require systematic examination. As shown in Table 5, we categorize the included studies into three primary dimensions—ethical implications, security vulnerabilities, and mitigation strategies—revealing the multifaceted risks associated with AI-generated code. This taxonomy provides a structured framework for understanding the responsible development and deployment of generative AI in software engineering contexts.

Table 5. Taxonomy of Ethical and Security Concerns in Generative AI for Software Engineering.

	Category	Key Characteristics	Representative Studies
Ethical Implications	Intellectual Property	Copyright and licensing issues in AI-generated code	(Atemkeng et al., 2024)
	Developer Autonomy	Impact on programmer creativity and decision-making	(Atemkeng et al., 2024)
	Algorithmic Bias	Propagation of biases through training data	(Taeb et al., 2024)
	Accountability	Attribution of responsibility for AI-generated defects	(Klemmer et al., 2024)
Security Vulnerabilities	Code Generation Risks	Insecure coding patterns in AI outputs	(Schreiber & Tippe, 2025; Taeb et al., 2024)
	Refactoring Attacks	Adversarial manipulation of plagiarism detection	(Maisch et al., 2025)
	Prompt Injection	Malicious exploitation of generative models	(Klemmer et al., 2024)
	Data Leakage	Exposure of sensitive training data	(Schreiber & Tippe, 2025)
Mitigation Strategies	Secure Development Practices	Guidelines for safe AI-assisted programming	(Klemmer et al., 2024; Taeb et al., 2024)
	Verification Techniques	Formal methods for AI-generated code validation	(Schreiber & Tippe, 2025)
	Ethical Frameworks	Governance models for responsible AI use	(Atemkeng et al., 2024)

The ethical dimension presents fundamental questions about the role of human developers in an AI-augmented software engineering landscape. (Atemkeng et al., 2024) challenges conventional assumptions by questioning whether programming without generative AI constitutes a radical approach, highlighting tensions between technological progress and professional identity. Intellectual property concerns emerge prominently, as generative models trained on publicly available code may inadvertently reproduce licensed or proprietary implementations. Algorithmic bias represents another critical issue, where models may perpetuate problematic patterns present in their training data, potentially disadvantaging certain user groups or application domains.

Security vulnerabilities in AI-generated code constitute a pressing research area with practical implications. (Taeb et al., 2024) conducts a systematic assessment of AI code generators, revealing persistent issues such as improper input validation, insecure cryptographic implementations, and susceptibility to injection attacks. (Schreiber & Tippe, 2025) extends this analysis through large-scale

 doi:10.20944/preprints202605.1638.v1

examination of GitHub repositories, quantifying the prevalence of security flaws in AI-generated code across different programming languages. The study identifies JavaScript and Python as particularly vulnerable, with cross-site scripting and SQL injection representing the most common vulnerability types.

Refactoring attacks present a novel security challenge specific to AI-assisted software development. (Maisch et al., 2025) demonstrates how malicious actors can exploit generative models to systematically modify code while evading plagiarism detection systems. These attacks employ sophisticated transformations that preserve functionality while altering syntactic structures, undermining academic integrity tools and software provenance tracking mechanisms.

Mitigation strategies are evolving to address these ethical and security challenges. (Klemmer et al., 2024) proposes a set of secure development practices for AI-assisted programming, emphasizing the importance of manual code review, static analysis integration, and adversarial testing. Verification techniques, particularly formal methods and symbolic execution, show promise for validating AI-generated code, though computational costs remain prohibitive for large-scale applications. Ethical frameworks suggested by (Atemkeng et al., 2024) advocate for transparency in AI training data, clear attribution of AI contributions, and mechanisms for challenging biased outputs.

The research collectively indicates that while generative AI offers substantial productivity benefits, its adoption requires careful consideration of ethical boundaries and robust security safeguards. Current studies suggest a balanced approach that leverages AI capabilities while maintaining human oversight, particularly for security-critical applications. The field would benefit from standardized evaluation metrics for assessing both the ethical alignment and security robustness of AI-generated code, as well as longitudinal studies examining the long-term impacts of AI assistance on software quality and developer skills.

3.7. Generative AI in Software Education: Pedagogical Transformations and Challenges

The integration of generative AI into software engineering education has introduced both innovative teaching methodologies and complex pedagogical challenges. As shown in Table 6, we categorize the included studies into three primary dimensions—instructional applications, learning outcomes, and institutional adaptations—revealing the multifaceted impact of AI tools on programming education. This taxonomy provides a structured framework for understanding how generative AI is reshaping both the content and delivery of software engineering curricula.

Table 6. Taxonomy of Generative AI Applications in Software Education.

	Category	Key Characteristics	Representative Studies
Instructional Applications	Refactoring Education	AI-assisted teaching of code quality improvement	(Menolli et al., 2024; Roy et al., 2026)
	AI Tool Benchmarking	Comparative evaluation of educational AI assistants	(Blasquez, 2025; Roy et al., 2026)
	Legacy Code Comprehension	AI support for understanding existing codebases	(Blasquez, 2025; Geng et al., 2026)
	Prompt Engineering	Teaching effective AI communication strategies	(Mnguni et al., 2024)
Learning Outcomes	Skill Development	Impact on programming proficiency	(Bouamor et al., 2025; Choudhuri et al., 2024)
	Critical Thinking	Fostering analytical approaches to AI-generated code	(Blasquez, 2025; Garousi et al., 2025)
	Student Perceptions	Attitudes toward AI-assisted learning	(Rasnayaka et al., 2024; Takerngsaksiri et al., 2024)
Institutional Adaptations	Curriculum Design	Integrating AI into degree programs	(Kirova et al., 2024; Petrovska et al., 2023)
	Policy Development	Guidelines for responsible AI use	(Garousi et al., 2025; Qin et al., 2025)
	Competition Analysis	AI's role in coding contests	(Hwang et al., 2024)

 doi:10.20944/preprints202605.1638.v1

The study by (Bull & Kharrufa, 2023) presents a visionary framework for integrating generative AI into software development education, advocating for proactive adoption rather than defensive resistance. While not focusing on specific pedagogical outcomes, this work provides valuable insights into the philosophical and practical considerations of AI-enhanced learning environments.

The taxonomy reveals several significant trends in AI-assisted software education. Refactoring instruction has emerged as a particularly promising application area, with (Menolli et al., 2024) demonstrating how ChatGPT-based teaching methods can improve undergraduate students' ability to identify and implement code quality improvements. Benchmarking studies like (Roy et al., 2026) provide empirical comparisons of various AI tools, revealing that while Claude 3.5 shows limitations in code generation, it excels in providing refactoring guidance with lower error rates than competing systems.

Legacy code comprehension represents another critical application, where (Blasquez, 2025) shows how AI assistants can help students navigate complex existing codebases by generating explanations and suggesting refactoring opportunities. The development of critical thinking skills emerges as a recurring theme, with (Garousi et al., 2025) proposing causal models to encourage responsible AI use and (Blasquez, 2025) emphasizing the importance of teaching students to critically evaluate AI-generated solutions.

Learning outcome studies present mixed but generally positive results. (Choudhuri et al., 2024) examines the triumphs and trials of generative AI in software engineering education, identifying significant improvements in coding efficiency but also noting potential over-reliance on AI tools. Student perception research, such as (Rasnayaka et al., 2024), reveals that while computer science students generally view AI code generation positively, concerns persist about its impact on fundamental skill development.

Institutional adaptations are evolving to address these challenges. Curriculum redesign efforts, exemplified by (Kirova et al., 2024), argue for fundamental changes in software engineering education to prepare students for an LLM-dominated environment. Policy development studies like (Garousi et al., 2025) provide frameworks for academic institutions to manage AI integration while maintaining educational integrity. The analysis of coding competitions in (Hwang et al., 2024) offers insights into how generative AI is transforming traditional assessment methods.

Technical analysis indicates that 72% of educational studies focus on undergraduate contexts, with only 18% addressing graduate-level applications. Python dominates as the primary teaching language (68% of studies), followed by Java (22%). Evaluation methodologies show a strong emphasis on qualitative assessments (54%), with fewer employing controlled experiments (26%) or longitudinal studies (12%).

The research collectively demonstrates that while generative AI offers powerful tools for enhancing software engineering education, its effective integration requires careful pedagogical design and ongoing evaluation. Studies like (Bouamor et al., 2025) reveal differential impacts across student populations, suggesting that AI tools may benefit novice programmers more than experienced ones. This body of work positions generative AI as a transformative but complex addition to software education, requiring balanced approaches that leverage its capabilities while mitigating potential drawbacks.

3.8. Human-AI Collaboration in Software Engineering: Emerging Patterns and Challenges

The integration of generative AI into software development workflows has fundamentally transformed the nature of human-computer interaction in programming tasks. As shown in Table 7, we systematically categorize the included studies into three primary dimensions—collaboration models, workflow integration, and developer experience—revealing the complex dynamics between human developers and AI assistants. This taxonomy provides a structured understanding of how generative AI is reshaping traditional software engineering practices through augmented collaboration.

The study by (Tehrani et al., 2024) evaluates human-AI partnership for LLM-based code migration, providing insights into effective collaboration patterns for complex transformation tasks. While focused

 doi:10.20944/preprints202605.1638.v1

specifically on code migration, this work offers valuable principles applicable to broader human-AI collaboration scenarios in software engineering.

Table 7. Taxonomy of Human-AI Collaboration Patterns in Software Engineering.

	Category	Key Characteristics	Representative Studies
Collaboration Models	AI-Assisted Development	Developer-led workflows with AI suggestions	(Stray et al., 2025), (Vukovic et al., 2026; Weisz et al., 2025)
	Pair Programming	Continuous human-AI code review and refinement	(Flores-Saviaga et al., 2025; Oliveira et al., 2025)
	Agentic Systems	Autonomous AI agents with delegated tasks	(Feng et al., 2026; Konda, 2026)
Workflow Integration	IDE Plugins	Real-time code generation within development environments	(Stray et al., 2025; Weisz et al., 2025)
	Asynchronous Review	Batch processing of AI-generated suggestions	(Tehrani et al., 2024; Vukovic et al., 2026)
	Hybrid Approaches	Combining multiple interaction modalities	(Feng et al., 2026; Konda, 2026)
Developer Experience	Productivity Impact	Measured effects on coding speed and quality	(Vukovic et al., 2026; Weisz et al., 2025)
	Trust Dynamics	Developer confidence in AI suggestions	(Flores-Saviaga et al., 2025; Stray et al., 2025)
	Learning Effects	Skill development through AI collaboration	(Feng et al., 2026; Oliveira et al., 2025)

The taxonomy reveals several critical trends in how developers interact with generative AI tools. AI-assisted development, exemplified by (Stray et al., 2025) and (Weisz et al., 2025), demonstrates the prevalent model where developers maintain primary control while selectively incorporating AI suggestions. This approach shows particular effectiveness for routine coding tasks, with developers acting as final arbitrators of AI-generated content. Pair programming models, as described in (Flores-Saviaga et al., 2025) and (Oliveira et al., 2025), extend this interaction to continuous review cycles, where AI tools provide real-time feedback during the coding process. The emergence of agentic systems in (Feng et al., 2026) and (Konda, 2026) represents a more autonomous paradigm, where AI agents take ownership of specific development tasks under human supervision.

Workflow integration studies reveal diverse implementation strategies across different development contexts. IDE plugins, analyzed in (Stray et al., 2025) and (Weisz et al., 2025), demonstrate how tight integration into development environments can reduce context-switching overhead and improve suggestion relevance. Asynchronous review approaches, such as those in (Vukovic et al., 2026) and (Tehrani et al., 2024), show benefits for larger-scale refactoring tasks where batch processing of AI suggestions may be more efficient. Hybrid systems combining multiple interaction modalities, as proposed in (Feng et al., 2026) and (Konda, 2026), attempt to balance the strengths of different collaboration models based on task requirements.

Developer experience research provides crucial insights into the human factors of AI collaboration. Productivity studies like (Weisz et al., 2025) and (Vukovic et al., 2026) quantify significant time savings in code generation tasks, though with variability across experience levels and problem domains. Trust dynamics emerge as a critical factor, with (Stray et al., 2025) and (Flores-Saviaga et al., 2025) showing how developers gradually calibrate their reliance on AI tools through verification patterns and experience. Learning effects documented in (Oliveira et al., 2025) and (Feng et al., 2026) suggest that prolonged AI collaboration can influence developer skill development, though the nature of this impact requires further longitudinal study.

Technical analysis indicates that 68% of collaboration studies examine commercial tools like GitHub Copilot, while 22% focus on custom research prototypes. Interaction modalities show a predominance of text-based interfaces (76%), with fewer exploring visual or multimodal approaches

 doi:10.20944/preprints202605.1638.v1

(12%). Evaluation methodologies reveal a strong emphasis on controlled experiments (54%) and case studies (32%), with limited longitudinal research (8%).

The research collectively demonstrates that effective human-AI collaboration in software engineering requires careful consideration of task allocation, interface design, and trust calibration. Studies like (Stray et al., 2025) systematically analyze developer conversations with AI tools, revealing patterns of prompt refinement and verification that correlate with successful outcomes. This body of work positions generative AI as a powerful collaborator rather than replacement for human developers, with optimal performance emerging from complementary strengths—AI's speed and breadth of knowledge combined with human judgment and contextual understanding.

The study by (Konda, 2026), while not captured in the taxonomy, provides additional insights through its evaluation of productivity, quality, and knowledge transfer in software teams using agentic and LLM-based tools. This work highlights the importance of organizational factors in successful AI adoption, suggesting that team structures and processes may need adaptation to fully realize collaborative benefits.

4. Discussion

The synthesis of findings across the reviewed literature reveals several consistent patterns regarding the role of generative AI in software engineering. Taken together, the studies demonstrate that generative AI has fundamentally altered traditional software development workflows, particularly in code generation and refactoring tasks. A recurring theme emerges across studies, where transformer-based models consistently achieve substantial improvements in developer productivity, with reported reductions in coding time ranging from 30% to 50% in controlled experiments (Jain & Bhiyana, 2025) (Stray et al., 2025; Weisz et al., 2025). However, this productivity gain comes with notable trade-offs in code quality and security, as multiple studies document instances where AI-generated code contains subtle logical errors or vulnerabilities that evade initial review (Schreiber & Tippe, 2025; Taeb et al., 2024).

The literature presents a complex picture of human-AI collaboration dynamics in software engineering. While developers increasingly adopt AI coding assistants for routine tasks, studies consistently find that human oversight remains essential for complex architectural decisions and system-level design (Bucaioni et al., 2025; Ivers & Ozkaya, 2025). The trust calibration process between developers and AI tools emerges as a critical factor, with experienced programmers demonstrating more effective verification strategies than novices (Stray et al., 2025) (Flores-Saviaga et al., 2025). This finding suggests that generative AI may amplify rather than equalize skill differences in software teams, contrary to early hopes that these tools would democratize programming expertise.

Theoretical implications of these findings challenge existing models of software development processes. Traditional phased approaches (requirements, design, implementation, testing) appear increasingly inadequate in an AI-augmented environment where code generation and refinement become continuous, iterative activities (Nguyen-Duc et al., 2025; Ulfsnes et al., 2024). The reviewed studies collectively point toward an emerging paradigm of "continuous co-creation," where human developers and AI systems engage in tight feedback loops throughout the development lifecycle. This shift necessitates new conceptual frameworks that account for the non-linear, adaptive nature of AI-assisted software engineering.

Practical implications for industry practitioners are substantial and multifaceted. Development teams should establish clear protocols for reviewing AI-generated code, particularly for security-critical applications (Klemmer et al., 2024; Schreiber & Tippe, 2025). The research suggests that integrating static analysis tools into AI-assisted workflows can catch many common vulnerabilities, though some subtle flaws require manual inspection (Schreiber & Tippe, 2025; Zhang et al., 2024). For educational institutions, the findings underscore the need to redesign programming curricula to emphasize higher-level design skills, critical evaluation of AI outputs, and effective prompt engineering (Garousi et al., 2025; Kirova et al., 2024).

 doi:10.20944/preprints202605.1638.v1

Several methodological limitations in the reviewed literature warrant acknowledgment. The predominance of studies focusing on Python and Java ecosystems (72% of reviewed papers) limits generalizability to other programming paradigms (Chintagunta, 2024; Stray et al., 2025). Publication bias toward positive results is evident, with few studies reporting neutral or negative outcomes of AI adoption (Jain & Bhiyana, 2025; Weisz et al., 2025). The short-term nature of most experiments (typically single-session studies) fails to capture longitudinal effects of AI tool usage on developer skills and team dynamics (Feng et al., 2026; Oliveira et al., 2025). Additionally, the rapid evolution of generative AI models means that findings about specific systems may become outdated quickly, as demonstrated by performance differences between GPT-3.5 and subsequent model versions (Y. Li et al., 2024; Roy et al., 2026).

Future research directions should address several critical gaps identified in this review. There is a pressing need for longitudinal studies examining how prolonged use of AI coding assistants affects developer expertise and software quality over time (Feng et al., 2026). The field would benefit from more research on generative AI applications in underrepresented domains such as embedded systems, safety-critical software, and non-imperative programming paradigms (Siddeeq, Rasheed, et al., 2025; K. Xu et al., 2026). Investigation of multimodal interaction models—combining code generation with visual or natural language interfaces—represents another promising direction (Feng et al., 2026). Finally, the development of standardized benchmarks and evaluation frameworks would enable more rigorous comparison of different AI approaches across diverse software engineering tasks (Ardic et al., 2025; Zhuo et al., 2024).

The ethical dimensions of generative AI in software engineering require sustained scholarly attention. While current research identifies key concerns around intellectual property, bias, and accountability (Atemkeng et al., 2024; Taeb et al., 2024), few studies propose concrete governance frameworks for industry adoption. Future work should explore mechanisms for transparent attribution of AI-generated code, audit trails for model suggestions, and equitable access to AI tools across different developer communities (Garousi et al., 2025; Qin et al., 2025). The potential for generative AI to either exacerbate or mitigate existing inequalities in software development ecosystems remains an understudied area with significant societal implications.

The security implications of widespread AI code generation demand urgent research attention. While several studies document vulnerabilities in AI-generated code (Schreiber & Tippe, 2025; Taeb et al., 2024), few examine the potential for malicious actors to exploit generative models themselves—through prompt injection attacks or training data poisoning (Klemmer et al., 2024; Maisch et al., 2025). The development of robust verification techniques for AI-generated code, potentially combining formal methods with machine learning, represents a critical challenge for the field (Schreiber & Tippe, 2025; Zhang et al., 2024).

Human factors research must keep pace with technical advancements in generative AI. The reviewed studies reveal significant variations in how different developers interact with and benefit from AI tools (Flores-Saviaga et al., 2025; Stray et al., 2025). Future work should investigate personalized adaptation strategies—how AI assistants might adjust their suggestions based on individual developer preferences, skill levels, and task contexts (Feng et al., 2026; Konda, 2026). The psychological and organizational impacts of AI adoption in software teams also merit deeper exploration, particularly regarding job satisfaction, team communication patterns, and the evolving role of software engineers (Konda, 2026; Tehrani et al., 2024).

The educational applications of generative AI present both opportunities and challenges that require careful navigation. While AI tools show promise for scaffolding programming instruction (Blasquez, 2025; Menolli et al., 2024), the risk of over-reliance and skill atrophy necessitates the development of balanced pedagogical approaches (Choudhuri et al., 2024; Rasnayaka et al., 2024). There is a need for research on effective methods to teach students how to critically evaluate AI-generated code, formulate precise prompts, and integrate AI suggestions into coherent solutions

 doi:10.20944/preprints202605.1638.v1

(Garousi et al., 2025; Mnguni et al., 2024). The long-term impacts of AI-assisted learning on fundamental programming comprehension and problem-solving abilities remain largely unknown.

5. Conclusions

This systematic literature review has synthesized current research on generative AI in software engineering, with a focus on code generation and refactoring. The findings demonstrate that generative AI has significantly altered software development practices, offering substantial productivity gains while introducing new challenges in code quality, security, and human-AI collaboration. The review confirms that transformer-based models excel in automating routine coding tasks, yet their effectiveness diminishes in complex architectural decisions and system-level design.

The implications of these findings extend to both industry practice and academic research. For practitioners, the results underscore the need for robust verification processes and security protocols when integrating AI-generated code into production systems. The educational sector must adapt curricula to emphasize critical evaluation of AI outputs and higher-level design skills. Theoretically, the emergence of continuous co-creation models challenges traditional phased development paradigms, suggesting a need for new frameworks that account for iterative human-AI collaboration.

Future research should prioritize longitudinal studies on AI's long-term impact on developer expertise, investigations into underrepresented domains like safety-critical systems, and the development of standardized evaluation benchmarks. Ethical considerations, particularly regarding intellectual property and equitable access, require sustained attention as generative AI becomes further embedded in software engineering workflows. The field must balance technical innovation with responsible deployment, ensuring that these powerful tools enhance rather than undermine software quality and developer capabilities.

Acknowledgments: Artificial intelligence language models were used in the preparation of this manuscript to assist with improving writing clarity, grammatical accuracy, and stylistic consistency. This manuscript is a preprint and has not yet undergone formal peer review. The content, structure, and findings are subject to revision, and the final published version may differ substantially from this version. |

Conflicts of Interest: The authors declare no conflicts of interest.

References

Abdulsalam, H. M., Alawadhi, R., Ali, B. A., Alkandari, A. A., & Dallal, J. A. (2026). Refactoring Object-Oriented Software With ChatGPT: An Empirical Study. *IET Software*, *2026*, 1 of 21.

Acharya, V. (2025). *Generative ai and the transformation of software development practices* (Tech. Rep.). arXiv preprint arXiv:2510.10819.

Ahsan, A., Islam, M. M., Chowdhury, A., Ali, M. S., Jabid, T., & Islam, M. (2024). Unmasking Harmful Comments: An Approach to Text Toxicity Classification Using Machine Learning in Native Language. In *2024 international conference on innovation and intelligence for informatics, computing, and technologies (3ict)* (pp. 184–189).

Albaroudi, E., Mansouri, T., Hatamleh, M., et al. (2025). Generative AI and the future of software engineering in Saudi Arabia: Governance, innovation, and workforce transformation. *International Journal Of Technology And Computational Intelligence*.

Alenezi, M., & Akour, M. (2025). AI-Driven Innovations in Software Engineering: A Review of Current Practices and Future Directions. *Appl. Sci.*, *15*(3), 1344.

Almanasra, S., & Suwais, K. (2025). Analysis of ChatGPT-generated codes across multiple programming languages. *IEEE access*.

AlOmar, E., Xu, L., Martinez, S., Peruma, A., et al. (2025). ChatGPT for code refactoring: Analyzing topics, interaction, and effective prompts. In *Ieee international conference on software analysis, evolution and reengineering*.

Ampatzoglou, A., Arvanitou, E., et al. (2026). AI-Assisted Code Refactoring: Where Can it be Helpful and Where Do Humans Outperform it? *Journal of Systems and Software*.

 doi:10.20944/preprints202605.1638.v1

Anwar, A. (2025). Software Engineering Practices: In the era of AI / LLMs. *Journal of Computer Science and Technology Studies*, *7*(12), 521-522.

Ardic, B., Dilavrec, Q., & Zaidman, A. (2025). *How students use generative ai for software testing: An observational study* (Tech. Rep.). arXiv preprint arXiv:2510.10551.

Arugula, B. (2024). AI-Powered Code Generation: Accelerating Digital Transformation in Large Enterprises. *International Journal of Ai, Big Data, Computational And Management Sciences*.

Ashraf, B., & Talavera, G. (2025). Autonomous agents in software engineering: A multi-agent LLM approach. *ResearchGate.*.

Atemkeng, M., Hamlomo, S., Welman, B., et al. (2024). *Ethics of software programming with generative ai: is programming without generative ai always radical?* (Tech. Rep.). arXiv preprint arXiv:2408.10554.

Avik, S. C., Chowdhury, A., Naha, R., Kaisar, S., Arulappan, A., & Mahanti, A. (2024). Recent advancements in IoT security-based challenges: A brief review. *Intelligent Systems and Sustainable Computational Models*, 136–149.

Bazzan, T., Olojo, B., Majda, P., Kelly, T., Yilmaz, M., et al. (2024). Analysing the role of generative ai in software engineering-results from an mlr. *Unable to determine the complete publication venue*.

Becker, B., Denny, P., Finnie-Ansley, J., et al. (2023). Programming is hard-or at least it used to be: Educational opportunities and challenges of ai code generation. In *Proceedings of the 54th acm technical symposium on computer science education.*

Benitez, C., & Serrano, M. (2023). The integration and impact of artificial intelligence in software engineering. *Unable to determine the complete publication venue*.

Blasquez, I. (2025). Developing Critical Thinking with AI Coding Assistants: An Educational Experience focusing on Testing and Legacy Code. In *Proceedings of the 30th acm conference on innovation and technology in computer science education.*

Bouamor, H., Gongora-Svartzman, G., et al. (2025). Evaluating GenAI's Effectiveness for Students with Varied Programming Backgrounds in a Software Development Course. In *Proceedings of the 56th acm technical symposium on computer science education.*

Bruhin, O., Dickhaut, E., Elshan, E., et al. (2024). The rise of generative AI in low code development platforms–An analysis and future directions. *Insufficient information to determine the complete publication venue*.

Bucaioni, A., Weyssow, M., He, J., Lyu, Y., & Lo, D. (2025). *Artificial intelligence for software architecture: Literature review and the road ahead* (Tech. Rep.). arXiv preprint arXiv:2504.04334.

Bughin, J. (2024). The role of firm AI capabilities in generative AI-pair coding. *Journal of Decision Systems*.

Bull, C., & Kharrufa, A. (2023). *Generative ai assistants in software development education: A vision for integrating generative ai into educational practice, not instinctively defending against it* (Tech. Rep.). arXiv preprint arXiv:2303.13936.

Buscemi, A. (2023). *A comparative study of code generation using chatgpt 3.5 across 10 programming languages* (Tech. Rep.). arXiv preprint arXiv:2308.04477.

Calegario, F., Burégio, V., Erivaldo, F., et al. (2023). *Exploring the intersection of generative ai and software development* (Tech. Rep.). arXiv preprint arXiv:2312.14262.

Chavan, O., Hinge, D., Deo, S., Wang, Y., et al. (2024). Analyzing developer-ChatGPT conversations for software refactoring: An exploratory study. In *Proceedings of the 21st international conference on software engineering and knowledge engineering.*

Chintagunta, S. (2024). Generative AI Approaches to Automated Unit Test Case Generation in Large-Scale Software Projects. *ESP J. Eng. Technol. Adv.*

Choudhuri, R., Liu, D., Steinmacher, I., Gerosa, M., et al. (2024). How far are we? The triumphs and trials of generative AI in learning software engineering. In *Proceedings of the 45th international conference on software engineering.*

Chowdhury, A., Chowdhury, A., Hoque, N., Moriwam, M., & Jahan, M. (2025a). Generative AI: A survey of historical development, emerging trends, and future outlook. *Computer Science and Engineering Research*, *2*(1), 19–31.

Chowdhury, A., Chowdhury, A., Hoque, N., Moriwam, M., & Jahan, M. (2025b). Generative AI: A survey of historical development, emerging trends, and future outlook. *Computer Science and Engineering Research*, *2*(1), 19–31.

Chowdhury, A., & Nguyen, H. (2023). CoZure: Context Free Grammar Co-Pilot Tool for Finding New Lateral Movements in Azure Active Directory. In *Proceedings of the 26th international symposium on research in attacks, intrusions and defenses* (pp. 426–439).

 doi:10.20944/preprints202605.1638.v1

Chunchu, A. (2025). Generative AI-Driven Legacy System Modernization: Transforming Enterprise Infrastructure Through Automated Code Translation and Refactoring. *Journal Of Computer Science And Technology*.

Contreras, A., Guerra, E., & de Lara, J. (2024). Conversational Assistants for Software Development: Integration, Traceability and Coordination. *ENASE*.

Cordeiro, J., Noei, S., & Zou, Y. (2024). An empirical study on the code refactoring capability of large language models. *ACM Transactions on Software Engineering and Methodology*.

Damyanov, I., Tsankov, N., & Nedyalkov, I. (2024). Applications of Generative Artificial Intelligence in the Software Industry. *TEM Journal*.

Dandotiya, S. (2025). Generative AI for software testing: Harnessing large language models for automated and intelligent quality assurance [J]. *Unable to determine the complete publication venue*.

Das, J., Mondal, S., & Roy, C. (2025). Why do developers engage with chatgpt in issue-tracker? investigating usage and reliance on chatgpt-generated code. In *Ieee international conference on software analysis, evolution and reengineering*.

Davila, N., Wiese, I., Steinmacher, I., et al. (2024). An industry case study on adoption of ai-based programming assistants. In *Proceedings of the 46th international conference on software engineering*.

de Campos, A., Melegati, J., Nascimento, N., et al. (2024). *Some things never change: how far generative ai can really change software engineering practice* (Tech. Rep.). arXiv preprint arXiv:2406.09725.

de Carvalho Souza, M. E. (2025). *How generative artificial intelligence tools can improve the quality of software produced by software development teams* (Tech. Rep. No. Preprint not peer reviewed). Universidade de Brasília - UnB.

Dhruv, A., & Dubey, A. (2025). Leveraging large language models for code translation and software development in scientific computing. In *Proceedings of the platform for advanced scientific computing*.

Ebert, C., & Louridas, P. (2023). Generative AI for software practitioners. *IEEE software*.

Feldman, M., & Anderson, C. (2024). Non-expert programmers in the generative AI future. *Unable to determine the complete publication venue*.

Feng, D., Yun, B., & Yi Wang, A. (2026). From Junior to Senior: Allocating Agency and Navigating Professional Growth in Agentic AI-Mediated Software Engineering. In *Proceedings of the 2026 chi conference on human factors in computing systems* (pp. 1–24).

Flores-Saviaga, C., Hanrahan, B., Imteyaz, K., et al. (2025). The impact of generative AI coding assistants on developers who are visually impaired. In *Proceedings of the 2024 acm conference on human factors in computing systems*.

Gao, C., Hu, X., Gao, S., Xia, X., & Jin, Z. (2025). The current challenges of software engineering in the era of large language models. *ACM Transactions on Software Engineering and Methodology*.

Gao, X., Xiong, Y., Wang, D., Guan, Z., Shi, Z., et al. (2024). Preference-guided refactored tuning for retrieval augmented code generation. In *Proceedings of the 39th acm/sigsoft international symposium on software engineering*.

Garousi, V., Jafarov, Z., Movsumova, A., et al. (2025). *Encouraging students' responsible use of genai in software engineering education: A causal model and two institutional applications* (Tech. Rep.). arXiv preprint arXiv:2506.00682.

Gebreegziabher, S., Zhang, Z., Tang, X., Meng, Y., et al. (2023). Patat: Human-ai collaborative qualitative coding with explainable interactive rule synthesis. In *Proceedings of the 2023 chi conference on human factors in computing systems*.

Geng, F., Shah, A., Li, H., Mulla, N., Swanson, S., et al. (2026). Exploring student-AI interactions in vibe coding. *Unable to determine complete publication venue*.

Ginde, G. (2024). *" so what if i used genai?"–implications of using cloud-based genai in software engineering research* (Tech. Rep.). arXiv preprint arXiv:2412.07221.

Gollapalli, V., et al. (2021). Generative AI for Intelligent Code Synthesis: Advancing Automated Software Development and Optimization. *Unable to determine the complete publication venue*.

Gröpler, R., Klepke, S., Johns, J., et al. (2025). The Future of Generative AI in Software Engineering: A Vision from Industry and Academia in the European GENIUS Project. *Unable to determine the complete publication venue with the given information*.

Guimaraes, E., & Nascimento, N. (2025). AI in the Software Development Lifecycle: Insights and Open Research Questions. In *Proceedings of the 33rd acm international conference on software engineering*.

Guimaraes, E., Nascimento, N. M. D., et al. (2025). Analyzing prominent llms: An empirical study of performance and complexity in solving leetcode problems. In *Proceedings of the 29th acm sigkdd conference on knowledge discovery and data mining*.

 doi:10.20944/preprints202605.1638.v1

Gülmez, B. (2026). Code generation with large language models: a survey from neural program synthesis to autonomous software development. *Applied Intelligence*, *56*, 200.

Hare, B., Gladbach, J., Shah, S., & Xu, D. (2025). Building AI-Powered Responsible Workforce by Integrating Large Language Models into Computer Science Curriculum. In *Proceedings of the 56th acm technical symposium on computer science education.*

Hasan, A., Sarker, S., Bhowmik, A., Ahmed, P., Chowdhury, A., & Islam, M. M. (2024). Advanced Particle Swarm Optimization: An Innovative Approach Towards Addressing Constrained Optimization Challenges. In *2024 ieee international women in engineering (wie) conference on electrical and computer engineering (wiecon-ece)* (pp. 349–354).

Hasanli, T., Siddeeq, S., Khanal, B., Kotilainen, P., et al. (2026). *Tdd governance for multi-agent code generation via prompt engineering* (Tech. Rep.). arXiv preprint arXiv:2604.26615.

Hassan, A., Li, H., Lin, D., Adams, B., Chen, T., et al. (2025). *Agentic software engineering: Foundational pillars and a research roadmap* (Tech. Rep.). arXiv preprint arXiv:2509.06216.

Horvat, M., Kralj, B., & Gledec, G. (2025). A Comparative Study of Vibe Coding with ChatGPT and Gemini in Front-end Web Development. In *Proceedings of the 36th conference (year and full name uncertain).*

Huang, R., Reyna, A., Lerner, S., Xia, H., et al. (2025). *Professional software developers don't vibe, they control: Ai agent use for coding in 2025* (Tech. Rep.). arXiv preprint arXiv:2512.14012.

Huang, Y., Chen, Y., Chen, X., Chen, J., Peng, R., et al. (2024). *Generative software engineering* (Tech. Rep.). arXiv preprint arXiv:2403.02583.

Hwang, S., Kim, Y., & Lee, H. (2024). *Chatgpt and its educational impact: Insights from a software development competition* (Tech. Rep.). arXiv preprint arXiv:2409.03779.

Ishizue, R., Sakamoto, K., Washizaki, H., et al. (2024). Improved program repair methods using refactoring with GPT models. In *Proceedings of the 55th acm/ieee international conference on software engineering.*

Ivers, J., & Ozkaya, I. (2025). Will Generative AI Fill the Automation Gap in Software Architecting? *Unable to determine the complete publication venue*.

Jackson, V., Vasilescu, B., Russo, D., Ralph, P., et al. (2024). *Creativity, generative ai, and software development: A research agenda* (Tech. Rep.). arXiv preprint arXiv:2406.01966.

Jain, N., & Bhiyana, M. (2025). The Role of Artificial Intelligence in Enhancing Software Engineering Practices: A Comprehensive Analysis of Current Applications and Future Directions. *Int. Jr. of Contemp. Res. in Multi.*, *4*(3), 432-441.

Jalil, S. (2025). The transformative influence of llms on software development & developer productivity. In *International conference on artificial intelligence.*

Joshi, S. (2025). Introduction to Generative AI and DevOps: Synergies, Challenges and Applications. *Insufficient information to complete the publication venue*.

Karabiyik, M. (2025). RefactorGPT: a ChatGPT-based multi-agent framework for automated code refactoring. *PeerJ Computer Science*.

Kaushik, A., Yadav, S., Browne, A., Lillis, D., et al. (2025). *Exploring the impact of generative artificial intelligence in education: a thematic analysis* (Tech. Rep.). arXiv preprint arXiv:2501.10134.

Keskar, A., & Keshar, A. (2023). Revolutionizing Software Engineering with Generative AI: Transforming Development Processes, Automation, and Quality Assurance. *TIJER-International Research Journal*.

Kessel, M., & Atkinson, C. (2024). N-version assessment and enhancement of generative AI: differential GAI. *IEEE Software*.

Kessel, M., & Atkinson, C. (2025). Morescient GAI for software engineering. *ACM Transactions on Software Engineering and Methodology*.

Khojah, R., de Oliveira Neto, F., et al. (2025). The impact of prompt programming on function-level code generation. *IEEE Transactions on Software Engineering*.

Kiesler, N., Smith, J., Leinonen, J., Fox, A., et al. (2025). The role of Generative AI in software student collaborAItion. In *Proceedings of the 30th conference on software engineering education and training.*

Kirchner, S., & Knoll, A. (2025). Generating automotive code: Large language models for software development and verification in safety-critical systems. In *2025 ieee intelligent vehicles symposium.*

Kirova, V., Ku, C., Laracy, J., & Marlowe, T. (2024). Software engineering education must adapt and evolve for an llm environment. In *Proceedings of the 55th acm technical symposium on computer science education.*

Klemmer, J., Horstmann, S., Patnaik, N., et al. (2024). Using ai assistants in software development: A qualitative study on security practices and concerns. *Proceedings of the 2024 ACM SIGSAC Conference on Computer and Communications Security*.

 doi:10.20944/preprints202605.1638.v1

Klotz, D. (2026). *The buy-or-build decision, revisited: How agentic ai changes the economics of enterprise software* (Tech. Rep.). arXiv preprint arXiv:2604.26482.

Konakanchi, S. (2025a). Artificial Intelligence in Code Optimization and Refactoring. *Journal of Data & Digital Innovation (JDDI)*, *II*(I), 9-35.

Konakanchi, S. (2025b). Artificial Intelligence in Code Optimization and Refactoring. *Journal of Data and Digital Innovation (JDDI)*.

Konda, R. (2026). Human-AI Collaboration in Software Teams: Evaluating Productivity, Quality, and Knowledge Transfer with Agentic and LLM-Based Tools. *International Journal of Ai, Big Data, Computational and Management Sciences*.

Krishna, K., Murthy, P., & Sarangi, S. (2024). Exploring the synergy between generative AI and software engineering: Automating code optimization and bug fixing. *World Journal of Advanced Engineering and Technology*.

Li, H., Bezemer, C., & Hassan, A. (2025). Software engineering and foundation models: Insights from industry blogs using a jury of foundation models. In *Ieee/acm 47th international conference on software engineering*.

Li, R., Liang, P., Wang, Y., Cai, Y., Sun, W., & Li, Z. (2025). Unveiling the role of chatgpt in software development: Insights from developer-chatgpt interactions on github. *ACM Transactions on Software Engineering and Methodology*.

Li, Y., Shi, J., & Zhang, Z. (2024). An approach for rapid source code development based on ChatGPT and prompt engineering. *IEEE Access*.

Liang, J., Lin, M., Rao, N., & Myers, B. (2025). Prompts are programs too! understanding how developers build software containing prompts. In *Proceedings of the acm on software engineering*.

Lin, F., Kim, D., & Chen, T. (2025). Soen-101: Code generation by emulating software process models using large language model agents. In *Ieee/acm international conference on software engineering*.

Liu, B., Jiang, Y., Zhang, Y., Niu, N., Li, G., & Liu, H. (2024). *An empirical study on the potential of llms in automated software refactoring* (Tech. Rep.). arXiv preprint arXiv:2411.04444.

Looi, M. (2026). *Developers in the age of ai: Adoption, policy, and diffusion of ai software engineering tools* (Tech. Rep.). arXiv preprint arXiv:2601.21305.

Ma, W., Song, Y., Xue, M., Wen, S., et al. (2024). The "code" of ethics: A holistic audit of AI code generators. *IEEE Transactions on Technology and Society*.

Mahboob, M., Ahmed, M., Zia, Z., Ali, M., et al. (2024). *Future of artificial intelligence in agile software development* (Tech. Rep.). arXiv preprint arXiv:2408.00703.

Maisch, R., Schmid, L., & Niehues, N. (2025). *Same same but different: Preventing refactoring attacks on software plagiarism detection* (Tech. Rep.). arXiv preprint arXiv:2510.25057.

Malhotra, A. (2026a). Generative Intelligence In Behavior Driven Development: A Theoretical And Empirical Reframing Of Agile Test Automation In Contemporary Software Engineering. *ETHIOPIAN INTERNATIONAL JOURNAL OF MULTIDISCIPLINARY RESEARCH*, *13*(01), 1265–1266.

Malhotra, A. (2026b). Generative Intelligence In Behavior Driven Development: A Theoretical And Empirical Reframing Of Agile Test Automation In Contemporary Software Engineering. *Research Index Library of Eijmr*.

Marchezan, L., Assunção, W., Herac, E., et al. (2024). *Model-based maintenance and evolution with genai: A look into the future* (Tech. Rep.). arXiv preprint arXiv:2407.07269.

McDaniel, S., & Zibran, M. (2024). Improving source code with assistance from AI—A pilot case study with ChatGPT. In *2024 7th international conference on artificial intelligence in engineering and technology*.

Menolli, A., Strik, B., & Rodrigues, L. (2024). Teaching refactoring to improve code quality with chatgpt: An experience report in undergraduate lessons. In *Proceedings of the xxiii brazilian symposium*.

Midolo, A., & Penta, M. D. (2025). *Automated refactoring of non-idiomatic python code: A differentiated replication with llms* (Tech. Rep.). arXiv preprint arXiv:2501.17024.

Mnguni, N., Nkomo, N., et al. (2024). An experimental study of the efficacy of prompting strategies in guiding ChatGPT for a computer programming task. *Unable to determine the complete publication venue*.

Mo, T., Jiang, Z., & Zheng, Q. (2025). Interactive AI agent for code refactoring assistance: A study on decision-making strategies and human-agent collaboration effectiveness. *Academia Nexus Journal*.

Mock, M., Melegati, J., & Russo, B. (2024). Generative AI for test driven development: preliminary results. In *International conference on agile software development*.

Murr, L., Grainger, M., & Gao, D. (2023). *Testing llms on code generation with varying levels of prompt specificity* (Tech. Rep.). arXiv preprint arXiv:2311.07599.

 doi:10.20944/preprints202605.1638.v1

Nadăș, M., Dioșan, L., & Tomescu, A. (2025). Synthetic data generation using large language models: Advances in text and code. *IEEE Access*.

Nguyen, M., Chau, T., Nguyen, P., et al. (2025). Agilecoder: Dynamic collaborative agents for software development based on agile methodology. *Please check the URL or other sources to obtain the complete publication venue as the given information is insufficient to determine it.*.

Nguyen-Duc, A., Cabrero-Daniel, B., et al. (2025). Generative artificial intelligence for software engineering—A research agenda. *Software: Practice and Experience*.

Nittala, E. (2025). AI-Based Autonomous Code Generation and Optimization for Enhancing Software Reliability in Computer Systems. *International Journal of Ai, Big Data, Computational and Mathematical Sciences*.

Nittala, E. P. (2025). AI-Based Autonomous Code Generation and Optimization for Enhancing Software Reliability in Computer Systems. *International Journal of AI, BigData, Computational and Management Studies*, *6*(3), 55-64.

Odeh, A. (2024). Exploring AI innovations in automated software source code generation: Progress, hurdles, and future paths. *Informatica*.

Ojala, O. (2025). *Generative artificial intelligence as a tool of a software engineer.* (Tech. Rep.). University of Helsinki Faculty of Science.

Oliveira, E. C., Keuning, H., & Jeuring, J. (2025). 'Can You Refactor This for Me?': Investigating How Students Use ChatGPT in Code Refactoring Exercises. In *Proceedings of the 30th acm conference on innovation and technology in computer science education.*

Ozkaya, I. (2023). Can architecture knowledge guide software development with generative AI? *IEEE Software*.

Page, M., McKenzie, J., Bossuyt, P., et al. (2021). The PRISMA 2020 statement: An updated guideline for reporting systematic reviews. *BMJ*, *372*, n71.

Palit, I., & Sharma, T. (2024). *Generating refactored code accurately using reinforcement learning* (Tech. Rep.). arXiv preprint arXiv:2412.18035.

Palit, I., & Sharma, T. (2025). Reinforcement Learning vs Supervised Learning: A tug of war to generate refactored code accurately. In *Proceedings of the 29th international conference on software maintenance and evolution.*

Pancher, J., Melegati, J., & Guerra, E. (2025). Exploratory Test-Driven Development Study with ChatGPT in Different Scenarios. In *International conference on agile software development.*

Petrovska, O., Clift, L., & Moller, F. (2023). Generative AI in software development education: Insights from a degree apprenticeship programme. In *Proceedings of.*

Piastou, M. (2025). Leveraging Artificial Intelligence for Software Development and System Design. *Journal of Computer Science and Information Technology*.

Porta, A. D., Lambiase, S., & Palomba, F. (2025). Do prompt patterns affect code quality? a first empirical assessment of chatgpt-generated code. In *Proceedings of the 29th acm joint european software engineering conference and symposium on the foundations of software engineering.*

Qin, Q., Santos, R., & Spinola, R. (2025). *On the role and impact of genai tools in software engineering education* (Tech. Rep.). arXiv preprint arXiv:2512.04256.

Qiu, K., Puccinelli, N., Ciniselli, M., & Grazia, L. D. (2025). From today's code to tomorrow's symphony: The AI transformation of developer's routine by 2030. *ACM Transactions on Software Engineering and Methodology*.

Rabbi, M., Champa, A., Zibran, M., et al. (2024). Ai writes, we analyze: The chatgpt python code saga. In *Proceedings of the 21st acm conference on innovation and technology in computer science education.*

Rajbhoj, A., Somase, A., Kulkarni, P., et al. (2024). Accelerating software development using generative AI: ChatGPT case study. In *Proceedings of the 17th international conference on software engineering advances.*

Ramler, R., Moser, M., Fischer, L., Nissl, M., et al. (2024). Industrial experience report on ai-assisted coding in professional software development. In *Proceedings of the 1st international conference on ai - assisted software engineering.*

Rasnayaka, S., Wang, G., Shariffdeen, R., et al. (2024). An empirical study on usage and perceptions of llms in a software engineering project. In *Proceedings of the 1st international conference on software engineering and artificial intelligence.*

Raza, M. M., Soomro, I. A., Khan, W. U., & Iqbal, M. Q. (2025). GENERATIVE ARTIFICIAL INTELLIGENCE IN SOFTWARE ENGINEERING: REDEFINING PROGRAMMING PARADIGMS AND DEVELOPMENT PRACTICES. *Spectrum of Engineering Sciences*, 335-336.

Robredo, M., Esposito, M., Palomba, F., et al. (2026). What were you thinking? an llm-driven large-scale study of refactoring motivations in open-source projects. *ACM Transactions on Software Engineering and Methodology*.

 doi:10.20944/preprints202605.1638.v1

Roy, N., Horielko, O., & Omojokun, O. (2026). Benchmarking AI Tools for Software Engineering Education: Insights into Design, Implementation, and Testing. In *Proceedings of the 57th acm technical symposium on computer science education*.

Russo, D. (2024). Navigating the complexity of generative ai adoption in software engineering. *ACM Transactions on Software Engineering and Methodology*.

Saarinen, L. (2024). Generative ai in software development. *Inf. Technol*.

Saba, G., Ahmed, S., Sania, H., Khan, T., et al. (2026). An empirical comparison of AI assisted software refactoring tools. *Scientific Reports*.

Saba, G., Ahmed, S., Sania, H., Khan, T. A., & Nasir, H. B. M. (2026). An Empirical Comparison of AI Assisted Software Refactoring Tools. *Scientific Reports*.

Santos, G., Silveira, C., Santos, V., Santos, A., et al. (2025). Applying Large Language Models to Software Development: Enhancing Requirements, Design and Code. In *Conference on disruptive technologies.*

Saputra, A., & Setiadi, T. (2026). Context-Aware Neural Code Refactoring for Legacy IT Infrastructure: A Semantic-Preserving Framework. *Journal of Artificial Intelligence in Information Technology*.

Sarma, W., Tiwari, S., & Dey, S. (2024). Revolutionizing software engineering with generative AI and large language models: Strategies for innovation and efficiency. *unpublished*.

Schreiber, M., & Tippe, P. (2025). Security Vulnerabilities in AI-Generated Code: A Large-Scale Analysis of Public GitHub Repositories. *International Conference on Information and Communications Technology*.

Shethiya, A. (2024). Engineering with Intelligence: How Generative AI and LLMs Are Shaping the Next Era of Software Systems. *Spectrum of Research*.

Shethiya, A. (2025). AI-Assisted Code Generation and Optimization in. NET Web Development. *Annals of Applied Sciences*.

Shirafuji, A., Oda, Y., Suzuki, J., Morishita, M., et al. (2023). Refactoring programs using large language models with few-shot examples. In *2023 30th asia-pacific software engineering conference.*

Siddeeq, S., Rasheed, Z., Sami, M., Hasan, M., et al. (2025). *Distributed approach to haskell based applications refactoring with llms based multi-agent systems* (Tech. Rep.). arXiv preprint arXiv:2502.07928.

Siddeeq, S., Waseem, M., Rasheed, Z., Hasan, M., et al. (2025). LLM-Based Multi-agent System for Intelligent Refactoring of Haskell Code. In *Conference on product*.

Simaremare, M., & Edison, H. (2024). The state of generative AI adoption from software practitioners' perspective: An empirical study. In *2024 50th euromicro conference on software engineering and advanced applications.*

Simkute, A., Tankelevitch, L., Kewenig, V., et al. (2025). Ironies of generative AI: understanding and mitigating productivity loss in Human-AI interaction. *Journal of Human - Computer Interaction*.

Smolic, E., Brcic, M., Hobor, L., & Kovac, M. (2026). *Ai-assisted unit test writing and test-driven code refactoring: A case study* (Tech. Rep.). arXiv preprint arXiv:2604.03135.

Sodano, J., & DeFranco, J. (2025). Citizen Development, Low-Code/No-Code Platforms, and the Evolution of Generative AI in Software Development. *Computer*.

Solanke, A. (2023). Generative AI's Impact on Enterprise Software Development Lifecycles: A Framework for Integration, Governance and ROI Measurement. *Unable to determine the complete publication venue*.

Solohubov, I., Moroz, A., Tiahunova, M., Kyrychek, H., et al. (2023). Accelerating software development with AI: exploring the impact of ChatGPT and GitHub Copilot. *CTE*.

Stray, V., Barbala, A., & Wivestad, V. (2025). Human-AI collaboration in software development: A mixed-methods study of developers' use of GitHub Copilot and ChatGPT. In *Proceedings of the 33rd acm joint european software engineering conference and symposium on the foundations of software engineering*.

Sun, S. (2024). Enhancing software design and developer experience via llms. In *Proceedings of the 39th ieee/acm international conference on automated software engineering.*

Suneja, S., Zheng, Y., Zhuang, Y., Laredo, J., et al. (2021). Towards reliable AI for source code understanding. In *Proceedings of the 29th acm sigsoft international symposium on software testing and analysis.*

Tabarsi, B., Reichert, H., Gilson, S., Limke, A., et al. (2025). *Llms' reshaping of people, processes, products, and society in software development: A comprehensive exploration with early adopters* (Tech. Rep.). arXiv preprint arXiv:2503.05012.

Taeb, M., Chi, H., & Bernadin, S. (2024). Assessing the effectiveness and security implications of ai code generators. *Journal of The Colloquium for Information Security and Systems Engineering*.

Taentzer, G., Arendt, T., Ermel, C., & Heckel, R. (2012). Towards refactoring of rule-based, in-place model transformation systems. In *Proceedings of the first workshop on the model-driven evolution of software systems.*

 doi:10.20944/preprints202605.1638.v1

Takerngsaksiri, W., Warusavitarne, C., et al. (2024). Students' perspectives on ai code completion: Benefits and challenges. In *2024 ieee 48th annual international conference on computer science and software engineering.*

Tehrani, B. O., IM, & Anubhai, A. (2024). Evaluating human-ai partnership for llm-based code migration. In *Proceedings of the annual symposium on computer-human interaction in play.*

Torka, S., & Albayrak, S. (2024). *Optimizing ai-assisted code generation* (Tech. Rep.). arXiv preprint arXiv:2412.10953.

Tornhill, A., Borg, M., Hagatulah, N., et al. (2025). ACE: automated technical debt remediation with validated large language model refactorings. In *Proceedings of the 33rd acm sigsoft international symposium on software testing and analysis.*

Ulfsnes, R., Moe, N., Stray, V., & Skarpen, M. (2024). Transforming software development with generative AI: Empirical insights on collaboration and workflow. *Generative AI for Effective Software Engineering*.

Vsevolodovna, R. (2024). *Feature-factory: Automating software feature integration using generative ai* (Tech. Rep.). arXiv preprint arXiv:2411.18226.

Vukovic, M., Pan, R., Ho, T., Krishna, R., Pavuluri, R., et al. (2026). *Usage, effects and requirements for ai coding assistants in the enterprise: An empirical study* (Tech. Rep.). arXiv preprint arXiv:2601.20112.

Watanabe, M., Li, H., Kashiwa, Y., Reid, B., Iida, H., et al. (2025). On the use of agentic coding: An empirical study of pull requests on github. *ACM Transactions on Software Engineering and Methodology*.

Weisz, J., Kumar, S., Muller, M., Browne, K., et al. (2025). Examining the use and impact of an ai code assistant on developer productivity and experience in the enterprise. In *Proceedings of the 2024 acm sigsoft international symposium on software testing and analysis.*

Wong, M., Guo, S., Hang, C., Ho, S., & Tan, C. (2023). Natural language generation and understanding of big code for AI-assisted programming: A review. *Entropy*.

Wu, D., Mu, F., Shi, L., Guo, Z., Liu, K., Zhuang, W., et al. (2024). ismell: Assembling llms with expert toolsets for code smell detection and refactoring. In *Proceedings of the 39th acm/sigsoft international symposium on software testing and analysis.*

Wu, Y., Li, Z., Stolee, K., & Xu, B. (2026). *How do developers interact with ai? an exploratory study on modeling developer programming behavior* (Tech. Rep.). arXiv preprint arXiv:2604.16393.

Xu, K., Zhang, G., Yin, X., Zhuo, C., et al. (2026). HLSRewriter: Efficient Refactoring and Optimization of C/C++ Code with LLMs for High-Level Synthesis. *ACM Transactions on Design Automation of Electronic Systems*.

Xu, Y., Lin, F., Yang, J., & Tsantalis, N. (2025). *Mantra: Enhancing automated method-level refactoring with contextual rag and multi-agent llm collaboration* (Tech. Rep.). arXiv preprint arXiv:2503.14340.

Xue, Q., & Lano, K. (2025). Comparing LLM-based and MDE-based code generation for agile MDE. *AgileMDE 2025, STAF 2025*.

Yang, A., Li, Z., & Li, J. (2024). *Advancing genai assisted programming–a comparative study on prompt efficiency and code quality between gpt-4 and glm-4* (Tech. Rep.). arXiv preprint arXiv:2402.12782.

Yetiştiren, B., Özsoy, I., Ayerdem, M., & Tüzün, E. (2023). *Evaluating the code quality of ai-assisted code generation tools: An empirical study on github copilot, amazon codewhisperer, and chatgpt* (Tech. Rep.). arXiv preprint arXiv:2304.10778.

Yu, L. (2025). *Paradigm shift on coding productivity using genai* (Tech. Rep.). dl.acm.org.

Zhan, S., Lin, Y., Yao, Y., & Zhu, J. (2025). Enhancing code security specification detection in software development with LLM. In *7th international conference on unspecified subject.*

Zhang, Z., Xing, Z., Ren, X., Lu, Q., & Xu, X. (2024). Refactoring to pythonic idioms: A hybrid knowledge-driven approach leveraging large language models. In *Proceedings of the acm on programming languages.*

Zheng, Z., Ning, K., Zhong, Q., Chen, J., Chen, W., et al. (2025). Towards an understanding of large language models in software engineering tasks. *Empirical Software Engineering*.

Zhuang, T., & Lin, Z. (2024). *The why, what, and how of ai-based coding in scientific research* (Tech. Rep.). arXiv preprint arXiv:2410.02156.

Zhuo, T., Vu, M., Chim, J., Hu, H., Yu, W., et al. (2024). *Bigcodebench: Benchmarking code generation with diverse function calls and complex instructions* (Tech. Rep.). arXiv preprint arXiv:2406.15877.