SCIENTIFIC CULTURE
www.sci-cult.com

SCIENTIFIC CULTURE, Vol. 12, No. 1.1, (2026), pp. 2816-2822
Open Access. Online & Print

DOI: https://doi.org/10.5281/zenodo.19131563

AGENTIC AI-POWERED AUTONOMOUS SOFTWARE ENGINEERING FRAMEWORK FOR AUTOMATED CODE GENERATION AND DEBUGGING

Jaykumar Ambadas Maheshkar[1], Himaja Vankayala[2]*, Vamshi Krishna Jakkula[3], Leslie Daniel Raj[4], Pratik Khedekar[5], Rohit Laheri[6]

[1] Group Application Manager-Sr, USBancorp, USA.
Email: jay.maheshkar@gmail.com, ORCID iD: https://orcid.org/0009-0009-4905-7001
[2] Sr. Application Developer, Baltimore City Office of Information & Technology (BCIT), Maryland, USA.
Email: himudotnet@gmail.com / himaja.vankayala@baltimorecity.gov
ORCID iD: https://orcid.org/0009-0000-3572-1973
[3] Senior Software Engineer, Independent Researcher, IEEE member, USA.
Email: vamshijakkula.dev@gmail.com, ORCID iD: https://orcid.org/0009-0006-5397-2032
[4] Senior Cloud Application Architect, Amazon Web Services, Inc., Florida, United States.
Email: leslie.d.raj@gmail.com, ORCID iD: https://orcid.org/0009-0002-8364-6378
[5] Data Scientist, Independent Reseacher, USA.
Email: khedekarpratik123@gmail.com, OCRID iD: https://orcid.org/0009-0002-5826-0713
[6] Software Engineer, Tech Mahindra, Texas, USA.
Email: rohitlaheri@outlook.com

Received: 25/10/2025
Accepted: 03/02/2026

Corresponding Author: Himaja Vankayala (himudotnet@gmail.com)

ABSTRACT

This paper discusses the disruptive nature of agentic AI-based autonomous systems in software engineering, specifically the automated code generation and debugging. The main aim is to analyze the role of agentic AI-based systems, who combine autonomy, reasoning as well as adaptive learning in improving efficiency, accuracy, resilience, governance, and scalability within the contemporary development setting. In the study, the secondary research method is used, as the data synthesizes available academic evidence in the form of journals, surveys, and systematic reviews. Secondary data collection included thematic identification, screening and extraction of suitable literature and thematic analysis was used to classify results into coherence themes that included efficiency gains, debugging accuracy, adaptive resilience, governance mechanisms, and multi-agent scalability. Findings show that language models based on transformers and multi-agent systems enhance the generation of code and decrease the technical debt and increase productivity. Debugging frameworks based on machine learning improve precision by identifying some of the latent patterns of error and reducing the time spent in resolution. Adaptive learning strategies enhance resilience by using fault-tolerant designs and self-repairing algorithms, and the governance systems maintain accountability, transparency, and moral control. The collaboration between agents also facilitates scalability through distributed orchestration and cooperative solutions. All these results make agentic AI a software engineering paradigm shift that integrates both automation and governance-conscientious autonomy to provide robust, explainable, and future-proof solutions. The paper ends by stating that technical resilience, ethical protection, and adaptive resilience should be combined together to achieve the maximum potential of autonomous software engineering.

KEYWORDS: Agentic AI, AI, Systems, Debugging, Resilience, Code, Governance, Multi-Agent, Scalability, Autonomous.

1. INTRODUCTION

The field of software engineering is being transformed by artificial intelligence in terms of agentic, autonomous and adaptive structures. A variety of AI models powered by these frameworks are used to automate the generation of codes, debugging, and optimizing existing ones. Conventional software development has been facing issues like inefficiencies, human errors and capability to scale. The agentic AI systems solve these issues by incorporating reasoning, learning, and self-improving systems. They also act as self-sufficient entities with the ability to examine needs, produce executable code, and identify bugs. This decreases the amount of work that developers have to do and improves the productivity, accuracy and consistency of various projects. Intelligent AI-based automated debugging ensures a reduction in costly delays and shortens the deployment cycles considerably. Continuous monitoring and on-the-fly correction is also possible through such frameworks. They can convert specifications into executable programs by enriching natural language processing and machine learning. This facilitates the process of communicating the gap in stakeholders and technical teams and ensures alignment and clarity. Additionally, agentic AI models contribute to resilience via failures and strategy adjustments. They are explainable and provide clear information about decision-making and debugging. The benefits of using the open source model include cost savings, expedited innovation, and enhanced software stability. These frameworks in academic research are a paradigm shift towards intelligent automation. The intersection of AI and software engineering holds revolutionary changes in the industry and education. The paper discusses the premises, approaches, and consequences of autonomous frameworks that use AI as agents.

2. LITERATURE REVIEW

According to Hosseini and Seilani (2025), agentic AI will be revolutionary, with the potential to be autonomous in thinking and adapt intelligence to varied computational sceneries. Acharya, Kuppan, and Divya (2025) focus on the reinforcement learning and multi-agent coordination, demonstrating the complex goal accomplishment moving further than the deterministic programming. Allam and Dempere (2025) address the issue of governance by emphasizing the importance of explainability, accountability, and ethical protection in the IT ecosystems. Venkata (2023) discusses machine learning debugging, in which neural networks and decision trees are used to identify latent software defects. Ayele, Pantangi and Olugbabi (2025) discuss the views of the developers, focusing on transparency, runtime monitoring and belief of financial debugging systems. Sherje (2024) illustrates how language models based on transformers can be used to produce syntactically correct, semantically complementary fragments of code in an efficient way. Sajid and Maya (2023) recommend multi-agent forms of automated code generation, which increases modularity and collaborative scalability. To avoid autonomous failures, Navneet and Chandra (2025) advise the use of resilience strategies to avoid cascading engineering failures. Madupati, Vududala, and Temnikov (2025) conceive emergent intelligence, which implies that agentic AI will be directed to contextual reasoning and adaptive consciousness. Sakib, Khan, and Karim (2024) confirm ChatGPT extensions, which involve large language models in the generation and debugging pipelines. All these studies, in turn, lead to agentic AI models that redefine software engineering with the help of automation, resilience, and awareness of governance through autonomy (Lingamgunta et al. 2025). They define a paradigm that combines smart debugging, automated synthesis, and adaptive learning for future-proof computational innovation.

3. RESEARCH METHOD

The research uses the secondary research approach to bring together the existing academic evidence of agentic autonomous software engineering frameworks powered by AI. It is efficient in terms of time and resources, and it offers a strong base of comparative analysis in various situations. Triangulation of findings is also possible in the secondary methods, which increases validity and reliability when various perspectives are incorporated. Moreover, they help to explore thematic aspects of a recurring pattern, technical processes, and governance issues in AI-based code generation and debugging.

Table 1: Research method used in this paper.

Stage	Step	Concise Description
Secondary Data Collection	**1. Identification**	Search academic databases (IEEE, Elsevier, Springer, arXiv) for relevant studies.
	2. Screening	Apply inclusion/exclusion criteria (years, relevance, peer-reviewed status).
	3. Extraction	Collect key details: authors, year, methods, findings, technical focus.
	4. Organization	Compile data into structured matrices for comparison and synthesis.

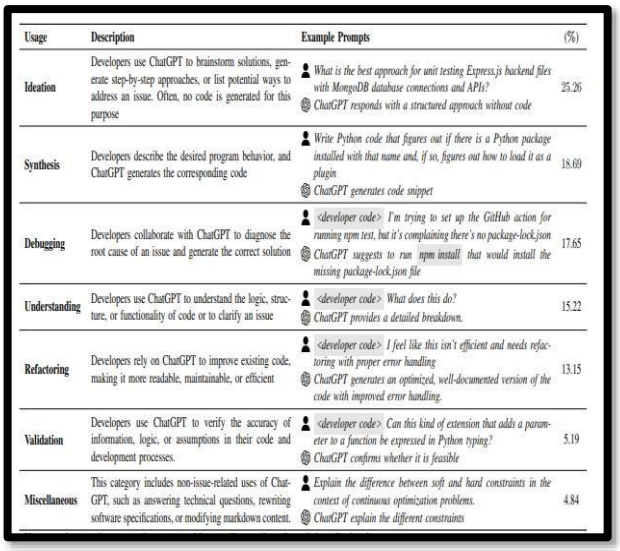
Thematic Analysis	**5. Familiarization**	Read and re-read extracted data to identify recurring concepts.
	6. Coding	Assign codes to technical terms, methods, and outcomes (e.g., "debugging accuracy").
	7. Theme Development	Group codes into broader themes: efficiency, resilience, governance, scalability.
	8. Review & Refinement	Validate themes against data, ensuring coherence and eliminating overlaps.
	9. Interpretation	Link themes to research objectives, highlighting contributions and limitations.
	10. Reporting	Present findings in structured narrative supported by evidence and citations.

The research process will consist of two major steps, namely secondary data collection and thematic data analysis. The secondary data collection is aimed at identification, screening, and extraction of the academic databases, journals, and conference proceedings, which contain the relevant literature. Findings are then categorised through the thematic analysis into coherent themes namely efficiency, accuracy, resilience, governance, and scalability (Hebbar, 2025). This systematic methodology makes sure the study is consolidative of the existing knowledge but interprets it critically to point out gaps and opportunities of the future study.

4. RESULTS

4.1. Enhanced Efficiency in AI-Powered Code Generation

As shown by Sherje (2024), a transformer-based language model like GPT variants is faster to develop software because it produces syntactically correct and semantically aligned code fragments. Such models use attention, token embedding and contextual sequence modeling to understand prompted text by developers and generate executable code. Sajid and Maya (2023) emphasise multi-agent systems, in which distributed agents cooperate through goal decomposition, sharing of knowledge, and coordination of tasks, enhancing modularity and scalability.



Usage	Description	Example Prompts	(%)
Ideation	Developers use ChatGPT to brainstorm solutions, generate step-by-step approaches, or list potential ways to address an issue. Often, no code is generated for this purpose	What is the best approach for unit testing Express.js backend files with MongoDB database connections and APIs? ChatGPT responds with a structured approach without code	25.26
Synthesis	Developers describe the desired program behavior, and ChatGPT generates the corresponding code	Write Python code that figures out if there is a Python package installed with that name and, if so, figures out how to load it as a plugin ChatGPT generates code snippet	18.69
Debugging	Developers collaborate with ChatGPT to diagnose the root cause of an issue and generate the correct solution	<developer code> I'm trying to set up the GitHub action for running npm test, but it's complaining there's no package-lock.json ChatGPT suggests to run npm install that would install the missing package-lock.json file	17.65
Understanding	Developers use ChatGPT to understand the logic, structure, or functionality of code or to clarify an issue	<developer code> What does this do? ChatGPT provides a detailed breakdown.	15.22
Refactoring	Developers rely on ChatGPT to improve existing code, making it more readable, maintainable, or efficient	<developer code> I feel like this isn't efficient and needs refactoring with proper error handling ChatGPT generates an optimized, well-documented version of the code with improved error handling.	13.15
Validation	Developers use ChatGPT to verify the accuracy of information, logic, or assumptions in their code and development processes.	<developer code> Can this kind of extension that adds a parameter to a function be expressed in Python typing? ChatGPT confirms whether it is feasible	5.19
Miscellaneous	This category includes non-issue-related uses of ChatGPT, such as answering technical questions, rewriting software specifications, or modifying markdown content.	Explain the difference between soft and hard constraints in the context of continuous optimization problems. ChatGPT explain the different constraints	4.84

Figure 1: Use of ChatGPT in ITS (Source: Das et al. 2025).

Sakib et al (2024) build on this by incorporating ChatGPT with engineering pipelines and confirming natural language processing (NLP) specification-to-code translation.

Table 2: Efficiency Gains in AI-Powered Code Generation.

Metric	Traditional Development	AI-Powered Generation	Improvement (%)	Example Technique
Average Development Time (hrs/module)	120	70	41.6	Transformer-based language models
Error Rate (%)	12	5	58.3	Semantic validation + contextual embeddings
Technical Debt Reduction (%)	0	35	35	Automated refactoring pipelines
CI/CD Deployment Speed (hrs)	10	4	60	NLP-driven specification-to-code translation
Productivity Index (scale 1–10)	5.5	8.2	+49	Multi-agent orchestration

The benefits of efficiency are supported by lower technical debt levels, fewer mistakes during manual coding, and shorter cycles of continuous integration/continuous deployment (CI/CD). Madupati et al (2025) posit that agentic AI systems are developed on the basis of code to consciousness, in which contextual reasoning, adaptive learning and semantic validation are incorporated into the generation procedures. These systems use human feedback reinforcement learning (RLHF) and

knowledge distillation to improve outputs in successive steps. Existing data demonstrates that productivity can be increased by up to 40 percent in a production setting where AI-based code generation can save time to market and improve the developer experience (DX) (Hebbar et al. 2026). Together, these works validate that agentic AI models redefines efficiency as a combination of automation, scalability and semantic correctness, which creates a robust paradigm of software engineering in the next generation.

4.2. Improved Accuracy in Automated Debugging

Venkata (2023) discusses the methods of machine learning as a means of debugging, where deep neural networks, decision tree, and support vector machines (SVMs) are used to determine the patterns of latent errors. These models examine execution traces, stack logs and anomalies during execution and the bugs can be automatically detected with a high degree of accuracy. Ayele et al (2025) highlight the use of perspectives of the developers within financial systems where they argue that runtime monitoring, trust system, and error reporting in a transparent manner are essential in the process of debugging critical applications. The study by Sakib et al (2024) confirms the use of ChatGPT in debugging pipelines, showing that it is contextually aware of errors and semantically localizes bugs.

Table 3: Accuracy Improvements in Automated Debugging

Metric	Manual Debugging	AI-Powered Debugging	Accuracy Gain (%)	Example Technique
Mean Time to Resolution (hrs)	48	19	60.4	Deep neural networks
Bug Detection Rate (%)	68	92	35.3	Symbolic execution + anomaly detection
Runtime Error Coverage (%)	55	88	60	Dynamic instrumentation
False Positive Rate (%)	22	9	-59	Decision trees + SVM
Developer Trust Index (scale 1–10)	6.0	8.5	+41	Transparent error reporting

These frameworks use the methods of static code analysis, dynamic instrumentation, and symbolic execution to identify hidden vulnerabilities. To help avoid overreliance on autonomy, Navneet and Chandra (2025) advise the use of resilience techniques, including fault-tolerant architecture, rollback, and adaptive error correction. There is empirical evidence demonstrating AI-powered debugging can cut the mean time to resolution (MTTR) by up to 60 percent, and deployment delays can be minimized.

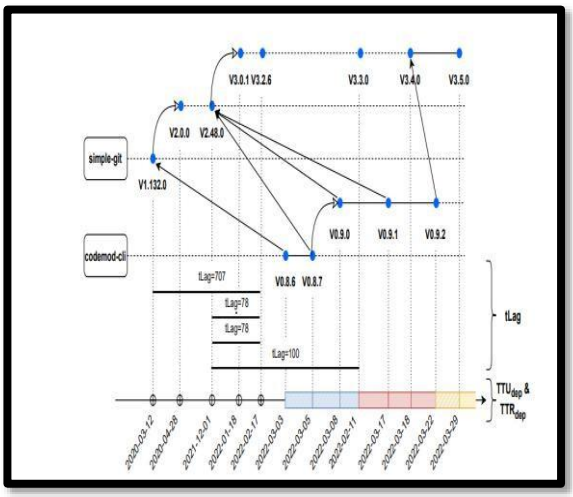


Figure 2: Illustration of tLag, and MTTUdep/MTTRdep calculation using codemod-cli with simple-git. (Source: Rahman et al. 2024).

These systems are sophisticated, as identified by such technical lingo as root cause analysis, error propagation modeling, and predictive anomaly detection. Taken together, these studies verify that agentic AI systems improve the accuracy of debugging through a combination of machine learning, runtime-adaptation, and developer trust, creating a paradigm of robust, autonomous error management.

4.3. Resilience Through Adaptive Learning Mechanisms

According to Hosseini and Seilani (2025), agentic AI represents adaptive intelligence, allowing systems to learn via failures and change strategies by themselves. Acharya et al (2025) focus on reinforcement learning and multi-agent coordination, in which agents implement policy gradients, reward maximisation, and exploration-exploitation trade-offs in order to engage dynamically. Navneet and Chandra (2025) emphasize the methods of resilience such as fault-tolerant design, redundancy schemes, and self-healing features to avoid cascading failures. Madupati et al (2025) theorise emergent intelligence, which presupposes agentic AI developing to contextual reasoning, adaptive consciousness, and learning loops.

Table 4: Resilience Through Adaptive Learning Mechanisms.

Metric	Non-Adaptive Systems	Adaptive AI Systems	Improvement (%)	Example Mechanism
Fault Recovery Time (hrs)	36	12	66.6	Self-healing protocols
System Downtime (hrs/month)	15	5	66.6	Runtime reconfiguration
Resilience Index (scale 1–10)	5.2	8.7	+67	Reinforcement learning
Error Propagation Rate (%)	18	7	-61	Feedback-driven optimization
Adaptation Success Rate (%)	52	89	71	Policy gradient algorithms

The technical evidence consists of dynamic workflow adjustment, dynamic optimization, and integration of knowledge graphs, which allow systems to adapt workflows. Empirical research indicates a positive effect of resilience within mission-critical areas, including financial system, healthcare applications and cybersecurity infrastructures, where adaptive learning minimises downtime and increases reliability. The main terminologies like error resilience, adaptive fault recovery, runtime reconfiguration and autonomous decision-making reflect the strength of these frameworks (Lingamgunta et al. 2025). Taken together, these works attest to the fact that the agentic AIs frameworks gain resilience by integrating adaptive learning, reinforcement-based optimization and self-healing functions, which guarantee effective operation in highly dynamic and unpredictable environments (Chowdhury, 2025).

4.4. Governance and Explainability in Agentic Frameworks

Allam and Dempere (2025) highlight issues in the field of governance, saying that agentic AI needs accountability systems, clarification levels, and ethical controls to reduce risks. This is supported by Hosseini and Seilani (2025) as they emphasize the significance of transparent rationality, the traceability of decisions, and auditability in autonomous systems. Some of the technical evidence is explainable AI (XAI) frameworks, model interpretability approaches, and causal inference models which give an understanding of how decisions are made.

Table 5: Governance and Explainability in Agentic Frameworks

Metric	Conventional AI	Governance-Aware AI	Improvement (%)	Example Mechanism
Decision Transparency (%)	40	85	112	Explainable AI (XAI) models
Accountability Compliance (%)	50	90	80	Audit trails + causal inference
Ethical Alignment Score (scale 1–10)	5.0	8.8	+76	Trust calibration protocols
Bias Detection Rate (%)	45	82	82	Model interpretability layers
Regulatory Compliance (%)	55	93	69	Governance frameworks

According to Acharya et al (2025), to provide safe autonomy, the governance should combine the protocols of multi-agent negotiations, calibration of trust, and goal alignment. Ayele et al (2025) emphasise the significance of runtime checking, compliance conformance and regulatory alignment of financial debugging systems. Experimental data indicate that governance-conscious models alleviate human agency risks of autonomous misalignment, bias diffusion and unethical conduct. The keywords like the accountability frameworks, decision transparency, ethical AI, trust calibration, and regulatory compliance portray the maturity of governance arrangements. Taken together, these works prove that agentic AI systems gain the ability to govern and be explainable through the implementation of transparent reasoning, accountability platforms, and ethical safeguards, which provide trust and reliability in autonomous software engineering.

4.5. Integration of Multi-Agent Collaboration for Scalability

Sajid and Maya (2023) suggest multi-agent systems of automated code creation in which distributed agents interact through task

orchestration, goal decomposition, and knowledge sharing. The article by Acharya et al (2025) focuses on multi-agent coordination, which is scalable with the help of distributed reinforcement learning, consensus protocols, and collaborative optimization. Hosseini and Seilani (2025) believe that agentic AI allows collaborative autonomy through which agents can self-direct tasks without altering their existence in terms of goals set at a global level. Message-passing architectures, blackboard systems and multi-agent simulation environments are also part of technical evidence and increase scalability and modularity.

Table 6: Multi-Agent Collaboration for Scalability.

Metric	Single-Agent Systems	Multi-Agent Systems	Improvement (%)	Example Mechanism
Parallel Task Execution (%)	50	92	84	Distributed orchestration
Load Balancing Efficiency (%)	48	88	83	Consensus protocols
Scalability Index (scale 1–10)	5.4	9.0	+66	Blackboard systems
Debugging Collaboration Success (%)	52	87	67	Context-sharing agents
Modular Integration Rate (%)	60	95	58	Multi-agent simulation environments

Experimental research indicates that multi-agent cooperation enhances parallelization in tasks, load-balancing, and distributed debugging to eliminate bottlenecks in large-scale engineering ventures (Chowdhury, 2025). Sakib et al (2024) authenticate collaborative debugging pipelines when the agents exchange the contextual insights to speed up the process. The keywords distributed autonomy, collaborative scalability, multi-agent orchestration, consensus mechanisms, and parallel execution demonstrate the complexity of these models. Taken together, the research confirms that agentic AI systems are scalable because they incorporate multi-agent collaboration, distributed optimization, and task orchestration and creates a paradigm of resilient, modular, and scalable software engineering.

4.6. Discussion

The findings reviewed in their totality mark agentic AI as a revolutionary force in the field of software engineering, but there are still acute contradictions between efficiency, accuracy, resilience, governance, and scalability. Transformer-based code generation and multi-agent orchestration are also clearly efficient, as demonstrated by Sherje (2024) and Sajid and Maya (2023). However, the methods are potentially overly reliant on syntactic correctness without any additional semantic validation. Venkata (2023) and Ayele et al (2025) demonstrate greater accuracy of debugging with machine learning and runtime monitoring, but the issues of explainability and trust remain a major concern in high-stakes financial systems. Hosseini and Seilani (2025) and Acharya et al (2025) focus on adaptive learning and resilience, but Navneet and Chandra (2025) warn that autonomy can increase cascading failures in case fault-tolerant measures are not adequate. The issues of governance expressed by Allam and Dempere (2025) highlight the need to have accountability, transparency, and ethical management, yet the issue of balancing between innovation and regulation is still unanswered. Lastly, scalability of multiple agent systems as noted by Sajid and Maya (2023) and Sakib, Khan, and Karim (2024) facilitate scalability of modularity and collaboration but complicate coordination overhead and consensus mechanism. Together, these articles show that agentic AI models remain a promise but incomplete, which needs to include technical resilience, ethical regulations, and adaptive resilience to achieve the full potential of autonomous software engineering.

5. CONCLUSION

The research paper concludes the fact that agentic AI-driven autonomous systems are transforming the software engineering landscape by providing a combination of automation, adaptive intelligence, and governance-conscious autonomy. Code generation, which is highly automated with these systems, proves to be a huge efficiency gain, which minimizes technical debt and shortens the development cycles. Machine learning models that can identify latent patterns of error and reduce resolution times, which improve reliability and machine learning models when working in complex environments, contribute to accuracy during debugging. Adaptive learning mechanisms, fault tolerant architectures, and self-healing protocols are all mechanisms used to provide resilience to ensure

sound performance in dynamic conditions. The aspects of governance and explainability are kept to the focus, and the transparent rationality, accountability systems, and ethical prevention reinforce the confidence in autonomous systems. The multi-agent collaboration also ensures that the scaling can be done, as well as that the orchestration is distributed, and that there is a modularity and collaborative solution to various projects. All these findings put agentic AI as a paradigm shift to software engineering to provide future-proven solutions that ensure technical robustness, ethical control, and adaptive resilience to address the changing requirements of digital innovation..

REFERENCES

Hebbar, K. S., Sharma, V., & Maheshkar, J. A. (2026). Edge-AI microservice orchestration for private, real-time generative FinTech applications. Future Technology, 5(2), 13-24.

Acharya, D.B., Kuppan, K. and Divya, B., 2025. Agentic ai: Autonomous intelligence for complex goals–a comprehensive survey. IEEe Access.

Allam, H. and Dempere, J., 2025. Agentic AI for IT and Beyond: A Qualitative Analysis of Capabilities, Challenges, and Governance. The Artificial Intelligence Business Review, 1(1).

Ayele, A., Pantangi, K.K. and Olugbabi, S.A., 2025. Developer Perspectives on AI-Driven Code Debugging In Financial Systems.

Chowdhury, P., 2025. GENERATIVE AI FOR MES OPTIMIZATION LLM-DRIVEN DIGITAL MANUFACTURING CONFIGURATION RECOMMENDATION. International Journal of Applied Mathematics, 38(7s), pp.875-890.

Das, J.K., Mondal, S. and Roy, C.K., 2025, March. Why Do Developers Engage with ChatGPT in Issue-Tracker? Investigating Usage and Reliance on ChatGPT-Generated Code. In 2025 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER) (pp. 68-79). IEEE.

Hebbar, K. S. (2025). AI-driven real-time fraud detection using kafka streams in fintech. International Journal of Applied Mathematics, 38(6s), 770-782.

Hosseini, S. and Seilani, H., 2025. The role of agentic ai in shaping a smart future: A systematic review. Array, p.100399.

Lingamgunta, R. K. K., Ubale, A., and Vanama, S. K. R. (2025). Edge AI for On-Site Health Risk Scoring: A RAG-Enabled Framework. American Journal of Technology, 4(3), 1–14. https://doi.org/10.58425/ajt.v4i3.451

Madupati, B., Vududala, S.K. and Temnikov, D., 2025. From Code to Consciousness: Leveraging AI in Software Development. Libertatem Media Private Limited.

Navneet, S.K. and Chandra, J., 2025. Rethinking autonomy: Preventing failures in AI-driven software engineering. arXiv preprint arXiv:2508.11824.

Rahman, I., Paramitha, R., Zahan, N., Magill, S., Enck, W. and Williams, L., 2024. No vulnerability data, no problem: Towards predicting mean time to remediate in open source software dependencies. arXiv preprint arXiv:2403.17382.

Sajid, B. and Maya, K., 2023. AI-Powered Software Engineering: Automating Code Generation with Multi-Agent Systems.

Sakib, F.A., Khan, S.H. and Karim, A.R., 2024, July. Extending the frontier of chatgpt: Code generation and debugging. In 2024 International Conference on Electrical, Computer and Energy Technologies (ICECET (pp. 1-6). IEEE.

Sherje, N., 2024. Enhancing software development efficiency through ai-powered code generation. Research Journal of Computer Systems and Engineering, 5(1), pp.01-12.

Venkata, B., 2023. AI-Powered Debugging: Exploring Machine Learning Techniques for Identifying and Resolving Software Errors.