International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 14, Issue 11, CONFLUENCE - 2026

An Ai-Driven Multi-Agent Framework for Automated Code Development and Analysis

Sinchana C[1], Sheetal S Kadam[2], Mary Stella F[3]
Student, Department of BCA, St. Francis College, Bengaluru[1]
Student, Department of BCA, St. Francis College, Bengaluru[2]
Assistant Professor, Department of BCA, St. Francis College, Bengaluru[3]
Email Id: mstella309@gmail.com

***Abstract*— Multi-Agent Coding Assistant is an AI-based application that makes use of multiple agents to assist users with numerous coding tasks. Programmers and developers usually perform various activities while developing a particular application or set of programs. These activities can include requirement analysis, actual coding, code analysis, debugging, testing, and code explanation. These tasks can be time-consuming and challenging, especially for inexperienced coders who may face numerous challenges when learning how to code or seeking to improve on their existing skills. The proposed solution aims to make the entire process simpler by introducing a multi-Agent framework in which individual agents address specific coding tasks. A Coordinator agent ensures that the entire process is overseen and that each request ends up in the right agent to achieve optimal results. Code Generation, Analysis, and Explanation, Debugging, and Testing agents address various tasks and offer better outcomes. The project's ultimate aim is to come up with an intelligent coding assistant that reduces the number of monotonous and time-consuming tasks that every coder has to deal with while building an application. In addition, it aims to resolve the problems that learners face when studying computer programming, including the need for explanations, debugging, and testing support. This way, the project will result in a viable prototype that will support developers while helping learners become more knowledgeable and skilled when it comes to computer programming.**

Keywords— Artificial Intelligence, Multi- Agent, Software Development, Conversional Interface.

I. INTRODUCTION

Building software means moving through many linked stages: gathering requirements, writing code, finding and fixing bugs, testing, reviewing, securing and maintaining the result. Each stage costs time and skilled effort, and mistakes made early tend to spread. Progress in machine learning, generative AI and large language models (LLMs) has produced tools that take over parts of this work [5], [6].

Current AI coding assistants can turn plain-language descriptions into source code, flag mistakes, recommend improvements, draft test scenarios and help trace faults [1], [15]. Most of them, though, cover just one stage, whether completion, review, security checking or testing, so developers end up switching between tools. The code they produce can also carry syntax faults, logical flaws, security weaknesses or inefficiencies, and must be checked and refined before use [1], [2].

This paper therefore presents an AI-Driven Framework for Automated Code Development and Analysis. Starting from a natural-language requirement, it chains together requirement interpretation, code generation, quality analysis, bug detection, security evaluation, test generation and code refinement. Analysis results are fed back to generation, so the final code is checked against the requirement and its expected behavior.

The design follows the multi-agent coding assistant concept: separate agents own separate development tasks and exchange information through a shared channel. The aim is to cut repeated manual effort, raise code quality, speed up debugging and testing, and give a smoother AI-assisted workflow. A person must still check and approve what the agents return.

The main contributions of this paper are:

- An integrated framework that combines requirement interpretation, code generation, analysis, debugging, testing, and verification in a single workflow.
- A multi-agent architecture consisting of Main (Coordinator), Coding, Review, and Test agents with an iterative verification loop.
- A conversational prototype evaluated on five representative programming tasks, including combined multi-agent requests.
- A review of related work and a discussion of limitations and future research directions.

II. BACKGROUND INFORMATION

This section presents background information on AI in software development, automated code generation and analysis, multi-agent systems, and automated testing, followed by the associated challenges and ethical considerations.

A. Artificial Intelligence in Software Development

AI now supports most stages of development, including coding, debugging, review, testing and optimization. Modern LLMs can write code from a natural-language brief, read

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 14, Issue 11, CONFLUENCE - 2026

existing programs, point out likely bugs and weak spots, and suggest improvements [5]. This removes much repetitive and error-prone effort. Yet the code they output may hold syntax or logic faults or settle for a weak solution, so generation should always be paired with analysis and checking.

B. Automated Code Generation and Analysis

Code generation translates a user's requirement into a program, and its quality hinges on how faithfully the requirement is understood and met. Analysis then examines the resulting code for faults, maintainability, coding-standard compliance and vulnerabilities. Static analysis inspects source code without running it, while dynamic analysis observes the program during execution and so exposes problems that appear only at run time. Using both covers more defect types than either alone.

C. Multi-Agent Systems for Software Engineering

In a multi-agent system, several agents cooperate toward a shared objective. For software engineering, agents can be assigned to requirement analysis, coding, review, test creation or error repair. Each agent completes its part and hands the outcome on, turning a large job into smaller linked ones. Dividing work among specialists is often easier than building one system that does everything, provided the agents communicate correctly, so collaboration and validation decide whether such a system succeeds.

D. Automated Testing and Iterative Code Refinement

Testing confirms that software behaves correctly and satisfies its requirements. Creating and running tests automatically allows a program to be examined against many inputs. Test outcomes, compiler warnings, exceptions and analysis findings guide corrections, after which the revised code is checked again. Passing compilation and tests does not prove the absence of faults, which is why well-chosen tests and clear validation criteria still matter.

E. Motivation for the Proposed Framework

Today's tools provide separate abilities such as generation, automated review, security analysis or test creation. Joining them in one cooperative process saves users from juggling tools and moving data between stages. The proposed framework merges requirement understanding, generation, quality analysis, bug and security checks, automated testing and iterative correction into a single development and validation cycle, and forms the conceptual basis of a multi-agent coding assistant.

F. Challenges and Threats

- **Code Correctness:** AI-generated code may contain logical errors, vulnerabilities, or suboptimal solutions and needs validation and quality-assurance checks [1].
- **Limited Context Understanding:** Current systems have difficulty with complex programming environments, project-specific contexts, and large, sophisticated codebases [2], [3], [4].
- **Security and Privacy:** Generated code may contain security flaws, and sharing proprietary code with AI services raises privacy concerns [6], [8].
- **Reliability and Evaluation:** Reliability, evaluation methods, and integration into real-world settings remain open issues for LLM-based software engineering [5].
- **Tool Fragmentation:** Separate tools for generation, review, testing, and debugging increase manual effort and data transfer between stages.

G. Ethical and Security Considerations

AI-assisted development raises issues of accountability, transparency and over-reliance on the tool [6]. Since generated code can be wrong or insecure, a human developer should stay responsible for reviewing and approving it [7]. Explanations also matter, because they help learners and developers see why a change was made [8]. Protecting user code and personal data is another concern, especially when requests are sent to external AI services.

III. LITERATURE REVIEW

This section looks at representative work on AI-driven software engineering. Advances in AI, generative AI and LLMs have improved code generation, debugging, testing, review, security analysis and maintenance, and researchers have automated one or several of these tasks. The studies are organized into code generation and software engineering, autonomous and multi-agent systems, and AI-driven testing, and are summarized in Table I.

A. AI-Based Code Generation and Software Engineering

Ghai et al. [1] examined AI for automatic code generation and found that it can turn developer requirements into code and save effort, although the snippets may hold logic errors and vulnerabilities, so validation and quality checks must accompany generation. Chen et al. [15] evaluated language models trained on code and showed that they can synthesize programs from natural-language descriptions, judged by functional-correctness tests. Shaheen et al. [2] applied cognitive models to generation, refactoring and bug fixing, but noted weak handling of complex environments and specific development contexts, which points to the need for context-aware systems.

Surveys give the wider picture. Hou et al. [5] reviewed a large body of work on LLMs across software engineering tasks, including generation, completion, testing, vulnerability detection, repair, review, requirements and maintenance, and flagged gaps in reliability, evaluation, context understanding and real-world integration. Alenezi and Akour [6] confirmed productivity gains and growing automation, while raising concerns about quality, security, privacy, explainability, ethics and dependence on the tools. Terragni et al. [7] considered how

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 14, Issue 11, CONFLUENCE - 2026

AI may reshape each phase from requirements to maintenance and stressed human-AI cooperation alongside reliability, security, validation and context awareness.

B. *Autonomous and Multi-Agent Development Systems*

Tufano et al. [3] introduced AutoDev, where AI agents operate directly on repositories to edit, retrieve, build, execute and test code. It shows what autonomous agents can do, yet it relies on the reasoning and context limits of the underlying language models, and large codebases remain hard. Sonkin and Tudose [4] proposed a workflow-centered framework for end-to-end generation covering context discovery, generation, testing, validation, error detection and self-healing, leaving scale, complexity and enterprise business logic unresolved.

Role-based multi-agent systems have also appeared: MetaGPT [16] gives cooperating agents human-like roles and standard procedures, ChatDev [17] runs role-playing agents through design, coding and testing phases, and AutoGen [18] offers a framework for applications built from conversing agents. Fedorov [11] described a decentralized multi-agent system for real-time security analysis that combines static and dynamic analysis, anomaly detection, code verification and architecture validation. Sharanarthi and Polineni [12] used several LLM agents for adaptive review, debugging and security analysis, showing that memory and user feedback limit repetition and improve suggestions. Manjusha et al. [13] built a collaborative environment with automated workflow management, and Manjula et al. [14] studied developers' coding patterns to give context-aware suggestions and spot inefficient code.

C. *AI-Driven Software Testing and Self-Healing*

Rehman et al. [8] looked at AI for code generation, testing and self-healing design; an ideal assistant would find and describe an error, propose and test a fix, and repeat, but such systems still need stronger cybersecurity, transparency, explainability and human oversight. Rajak [9] suggested using NLP and deep learning to turn requirements into context-aware, risk-prioritized test cases with feedback, though complex requirements still need people. Kathiresan [10] generated test cases with deep learning, reinforcement learning and evolutionary algorithms, which lowers manual effort and improves coverage and defect detection but depends on training-data quality and system complexity.

D. *Comparative Discussion*

Across these studies, AI assists almost every development stage, yet most work targets a single stage or full autonomy, and all of it stresses validation and human oversight. AutoDev [3] and the workflow framework of [4] aim at end-to-end automation; the proposed framework instead offers a conversational interface in which specialized agents handle generation, debugging, explanation, testing and improvement, which also helps learners. Against [12], which centers on review, debugging and security, it adds generation, explanation and testing in one interface, and against [13], which focuses on collaboration and process management, it supports the individual coder.

TABLE I
LITERATURE REVIEW OF REPRESENTATIVE STUDIES

#	Title	Authors	Year	Methods / Focus	Key Finding
1	AI in system and software engineering for auto code generation	Ghai et al.	2024	AI-based code generation	Saves effort, but output needs validation
2	AI-driven augmented software engineering	Shaheen et al.	2026	Cognitive models; generation, refactoring, bug fixing	Limited understanding of complex contexts
3	AutoDev: automated AI-driven development	Tufano et al.	2024	AI agents on repositories (edit, build, execute, test)	Autonomous agents feasible; large codebases are hard
4	Workflow-centric framework for end-to-end code generation	Sonkin & Tudose	2025	Context discovery, generation, testing, self-healing	Single workflow; scale and enterprise logic are open
5	LLMs for software engineering: systematic review	Hou et al.	2024	Systematic literature review	Wide applicability; reliability and evaluation gaps
6	AI-driven innovations in software engineering	Alenezi & Akour	2025	Review of practices and directions	Productivity gains; quality, security, ethics concerns
7	The future of AI-driven software engineering	Terragni et al.	2025	Roadmap of AI/LLM impact	Human-AI cooperation; validation is essential
8	AI in software engineering: generation, testing, self-healing	Rehman et al.	2026	Code generation, testing, self-healing	Needs security, transparency, human monitoring
9	AI-driven automated testing with NLP and deep learning	Rajak	2025	NLP, deep learning, risk-prioritized tests	Requirements to tests; human input still needed
10	Automated test case generation with AI	Kathiresan	2024	Deep learning, RL, evolutionary algorithms	Better coverage; depends on data quality

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 14, Issue 11, CONFLUENCE - 2026

#	Title	Authors	Year	Methods / Focus	Key Finding
11	Multi-agent system for real-time security analysis	Fedorov	2025	Multi-agent; static and dynamic analysis	Improved security decision-making
12	Multi-agent LLM for adaptive code review, debugging, security	Sharanarthi & Polineni	2025	Multi-agent LLM with memory and feedback	Avoids repetition; better suggestions
13	AI-powered collaborative development environment	Manjusha et al.	2026	Workflow management; collaboration	AI assistance for collaborative development
14	Developer pattern analysis and code suggestions with AI	Manjula et al.	2025	Pattern analysis; context-aware suggestions	Identifies inefficient code; suggests fixes
15	Evaluating LLMs trained on code	Chen et al.	2021	Code-trained LLM; functional-correctness evaluation	Programs synthesized from descriptions
16	MetaGPT: multi-agent collaborative framework	Hong et al.	2023	Role-based agents with standard procedures	Structured roles aid agent collaboration
17	ChatDev: communicative agents for software development	Qian et al.	2023	Role-playing agents across design, coding, testing	Chat-based multi-agent development
18	AutoGen: multi-agent conversation framework	Wu et al.	2023	Customizable conversable agents	Flexible multi-agent LLM applications

IV. METHODOLOGY

The proposed AI-powered multi-agent framework automates different aspects of code development and analysis using multiple AI agents. Unlike common approaches in which a single AI completes the whole process, the framework splits the process into distinct tasks and assigns each to a specific agent. Fig. 1 shows the architecture of the framework.

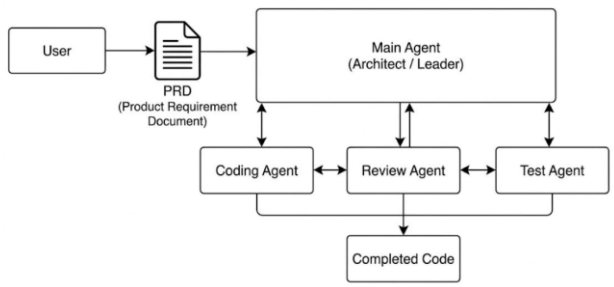


Fig. 1. Architecture of the proposed multi-agent framework

A. Agents and Their Roles

User Request. The user submits a requirement, shown in Fig. 1 as a Product Requirement Document (PRD). The first agent to receive and process it is the Main (Coordinator) Agent.

- **Main (Coordinator) Agent:** Interprets the requirement and passes the request to the agent that corresponds to the required action. It also verifies that everything has been completed to standard: code generated, reviewed, debugged, and tested. If the final test fails, the code is sent back for further debugging and testing. At the end, it returns the final code, explanation, errors, debugging, and testing results to the user through the user interface.
- **Coding Agent:** Responsible for developing the program code from the requirement. Once the code is generated, it is passed to the Review Agent.
- **Review Agent:** Reviews the code and highlights errors, bugs, logical issues, and other problems, including issues of format, syntax, and readability. It can also provide an explanation of the code.
- **Test Agent:** Handles the bugs and errors that were reported, providing corrected code, then runs tests on the new code to ensure that it works as intended. The result is sent back to the Main Agent.

B. Workflow

The workflow, shown in Fig. 2, is: User Requirement → Coordinator Agent → Code Generation → Code Analysis → Debugging → Testing → Verification → Final Output. The Coordinator verifies the test result. If verification fails, the code returns to debugging and testing; otherwise the final output is delivered. The iterative procedure is summarized below.

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 14, Issue 11, CONFLUENCE - 2026

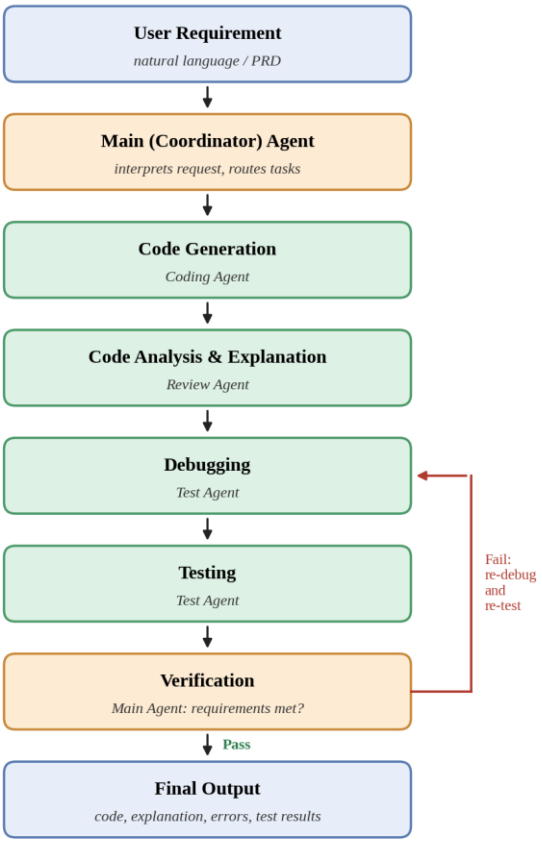


Fig. 2. Workflow of the framework with the verification feedback loop

- **Step 1:** The Main Agent receives and interprets the user requirement.
- **Step 2:** The Coding Agent generates code for the requirement.
- **Step 3:** The Review Agent analyzes the code and reports errors, bugs, and quality issues, and may explain the code.
- **Step 4:** The Test Agent debugs the code and runs tests.
- **Step 5:** The Main Agent verifies the result; on failure, Steps 4 and 5 are repeated, and on success the final output is returned to the user.

C. Main Components

Table II lists the main components of the framework and their functions.

TABLE II
MAIN COMPONENTS OF THE FRAMEWORK

Component	Function
User Interface	Accepts coding requirements and displays the final results
Coordinator Agent	Manages the whole coding process and controls the other agents
Code Generation Agent	Generates code according to the user requirements
Code Analysis Agent	Analyzes the code and highlights logical issues, bugs, and errors
Debugging Agent	Highlights and fixes errors, bugs, and logical issues
Testing Agent	Runs tests to make sure that all functions work correctly
Final Output Module	Provides the final output, explaining the results and giving code suggestions

D. Design Considerations

The design rests on three ideas. Each agent has a narrow duty, which keeps the system modular so agents can be improved or swapped independently. Agents pass results through a shared channel managed by the Main Agent, so one stage's output feeds the next. The user stays in the loop: final code, explanation and test results are shown for human checking, and a practical build should cap refinement iterations to prevent endless loops.

V. RESULTS AND DISCUSSION

A. Evaluation Methodology

The Multi-Agent Coding Assistant was tried on a range of programming tasks covering code generation, debugging, explanation, testing and improvement. Evaluation was conversational: the user described a task in natural language and the suitable agent was chosen to carry it out. Both single-task and multi-step requests needing several agents were included. Each response was judged for accuracy, relevance, clarity and whether the task was completed.

B. Test Cases

Table III lists the representative test cases and the expected results.

TABLE III
TEST CASES AND EXPECTED RESULTS

Test Case	Task	Expected Result
TC1	Create a Python program for prime numbers	Correct Python code
TC2	Create a C program for sum of numbers	Correct C code
TC3	Debug improper indentation	Corrected code
TC4	Explain sorting problem	Logic and explanation
TC5	Test a calculator program	Suitable input and output

C. Key Findings

- **Single Interface for Diverse Tasks:** The assistant addressed diverse programming-related demands through one interactive interface, and the specialized agents performed code generation, debugging, explanation, testing, and improvement.
- **Combined Requests:** The system processed inquiries that combined several tasks. For instance, when a user requested a program to be generated, explained, and tested, the appropriate agents performed each task and delivered the results in one integrated response.

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 14, Issue 11, CONFLUENCE - 2026

- **Systematized Coding Process:** Dividing the work among agents organizes the coding process, from writing and debugging code to explaining, testing, and improving it.

D. Limitations

- **Scale of Evaluation:** The evaluation covers a small set of representative tasks and was reviewed qualitatively. Larger benchmarks and quantitative metrics are needed to measure performance more rigorously.
- **Reliability of Generated Code:** As noted in the literature [1], [5], AI-generated code may still contain errors, so the verification loop does not remove the need for human review.
- **Scope:** The prototype handles single-file programming tasks in a limited set of languages and does not yet address large multi-file projects or enterprise-level logic [4].

E. Overall Outcome

The tests indicate that a single interface backed by several specialized AI agents can deliver varied programming services. The assistant structures the coding process, from writing and debugging to explaining, testing and improving code.

VI. CONCLUSION

The Multi-Agent Coding Assistant shows how several AI agents can jointly provide programming help. Bringing code generation, debugging, explanation, testing and quality improvement together makes it a handy, organized tool for students and developers, and the multi-agent approach handled the different kinds of request well. Although it can be developed further, it works as a prototype and a foundation for more capable assistants.

Future Work: The framework can be improved to provide more capable and reliable programming assistance, including better individual agent performance and collaboration among agents.

- **More Programming Languages:** Support for languages such as C++, JavaScript, PHP, and SQL.
- **Better Agent Collaboration:** Improved communication and coordination between agents.
- **Automatic Code Testing:** Automatic testing and verification of generated code.
- **Better Code Quality:** Suggestions for improving readability, performance, and security.
- **Larger Project Understanding:** Understanding and working with larger projects that involve multiple files.
- **IDE Integration:** Integration with coding tools and IDEs so that the assistant can be used while coding.
- **User Feedback:** Use of user feedback to improve the quality and usefulness of suggestions.
- **Better Security and Privacy:** Better protection of the user's code and personal information.

REFERENCES

[1] Ghai, A. S., Rawat, V., Gupta, V. K., & Ghai, K. (2024). Artificial Intelligence in System and Software Engineering for Auto Code Generation. 2024 International Conference on Electrical Electronics and Computing Technologies (ICEECT). https://ieeexplore.ieee.org/abstract/document/10738945/

[2] Shaheen, A., Al Khaldy, M., Alzyadat, W., & Alhroob, A. (2026). AI-driven augmented software engineering: Leveraging cognitive models for enhanced code generation. Journal of Computational and Cognitive Engineering, 5(2), 213-221. https://ojs.bonviewpress.com/index.php/JCCE/article/view/6123

[3] Tufano, M., Agarwal, A., Jang, J., Moghaddam, R. Z., & Sundaresan, N. (2024). AutoDev: Automated AI-driven development. arXiv preprint arXiv:2403.08299. https://arxiv.org/abs/2403.08299

[4] Sonkin, V., & Tudose, C. (2025). Beyond snippet assistance: A workflow-centric framework for end-to-end AI-driven code generation. Computers, 14(3), 94. https://www.mdpi.com/2073-431X/14/3/94

[5] Hou, X., Zhao, Y., Liu, Y., Yang, Z., Wang, K., Li, L., Luo, X., Lo, D., Grundy, J., & Wang, H. (2024). Large language models for software engineering: A systematic literature review. ACM Transactions on Software Engineering and Methodology, 33(8). https://doi.org/10.1145/3695988

[6] Alenezi, M., & Akour, M. (2025). AI-driven innovations in software engineering: A review of current practices and future directions. Applied Sciences, 15(3), 1344. https://www.mdpi.com/2076-3417/15/3/1344

[7] Terragni, V., Vella, A., Roop, P., & Blincoe, K. (2025). The future of AI-driven software engineering. ACM Transactions on Software Engineering and Methodology, 34(5). https://dl.acm.org/doi/10.1145/3715003

[8] Rehman, M. U., Khan, I., Javed, A., & Mir, V. (2026). Artificial intelligence in software engineering: Automated code generation, testing, and self-healing system design. Spectrum of Engineering Sciences, 4(5), 490-508. https://www.thesesjournal.com/index.php/1/article/view/2728

[9] Rajak, A. (2025). An AI-driven framework for automated software testing using natural language processing and deep learning.

[10] Kathiresan, G. (2024). Automated test case generation with AI: A novel framework for improving software quality and coverage. World Journal of Advanced Research and Reviews, 23(2), 2880-2889.

[11] Fedorov, S. (2025). AI-Driven Multi-Agent System for Real-Time Security Analysis of Software Releases. 2025 2nd International Conference on Intelligent Computing and Robotics (ICICR), 1195-1199. https://doi.org/10.1109/ICICR65456.2025.00212

[12] Sharanarthi, T., & Polineni, S. (2025). Multi-Agent LLM Collaboration for Adaptive Code Review, Debugging, and Security Analysis. 2025 International Conference on Mechatronics, Robotics, and Artificial Intelligence (MRAI), 541-546. https://doi.org/10.1109/MRAI65197.2025.11135756

[13] (2026). An AI-Powered Collaborative Development Environment with Automated Workflow Management. 2026 International Conference on Connected Intelligence for Industrial Applications (CI2A). https://doi.org/10.1109/CI2A69097.2026.11576837

[14] Manjula, G., Yashwanth, A., & Nitish, K. P. (2025). Automated Developer Pattern Analysis and Code Suggestions with AI. Indian Journal of Computer Science and Technology, 4(2), 21-31. https://doi.org/10.59256/indjcst.20250402002

[15] Chen, M., et al. (2021). Evaluating large language models trained on code. arXiv preprint arXiv:2107.03374. https://arxiv.org/abs/2107.03374

[16] Hong, S., et al. (2023). MetaGPT: Meta programming for a multi-agent collaborative framework. arXiv preprint arXiv:2308.00352. https://arxiv.org/abs/2308.00352

[17] Qian, C., et al. (2023). ChatDev: Communicative agents for software development. arXiv preprint arXiv:2307.07924. https://arxiv.org/abs/2307.07924

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 14, Issue 11, CONFLUENCE - 2026

[18] Wu, Q., et al. (2023). AutoGen: Enabling next-gen LLM applications via multi-agent conversation. arXiv preprint arXiv:2308.08155. https://arxiv.org/abs/2308.08155

(This work is licensed under a Creative Commons Attribution 4.0 International License.)