

AI-Assisted Programming, Memory Safety, Multi-Paradigm Design and Emerging Development Models

Nidhi Chahal^{1*}, Purshotam Jha², Sonia³, Vaishanvi⁴

Abstract

As the field of artificial intelligence (AI) continues to evolve at a fast pace, and specifically with the rise of large language models (LLMs), the software design, implementation, testing, and maintenance landscape has undergone unprecedented transformation. Although programming languages are still the main language for writing computational logic, the way developers interact with programming languages is now increasingly mediated by AI-based tools. Current AI coding systems can help with code generation, program explanation, error detection, test case generation, program translation between languages, and software documentation. This process is introducing a new type of programming that relies on AI to generate code, but with the human developer setting out requirements and ensuring the machine code meets them rather than writing all the lines. In this paper, we provide a selective overview of the recent advances in the field of programming languages and AI. It covers AI code generation, LLMs for software engineering, AI agents, natural language programming, memory-safe programming, and the evolving status of modern programming languages like Python, Rust, and TypeScript. There is significant evidence that shows that AI development tools are in use, but the problems of correctness, security, privacy, and verification are still substantial. The paper thus examines opportunities and challenges of AI-aided programming and suggests a human-in-the-loop approach to development that combines the process of generation, static analysis, testing, security validation, and human review. Finally, new research directions are explored, such as AI-native programming environments, verified code generation, agentic software development, and programming languages for reliable human–AI collaboration.

Keywords: Artificial intelligence, programming languages, large language models, AI-assisted programming, code generation, AI agents, software engineering, memory safety, human–AI collaboration

INTRODUCTION

Programming languages have continuously developed since the advent of computer hardware, software engineering, and application needs. Each step has added more abstraction, productivity, portability, and reliability to the various levels of machine and assembly languages, procedural, object-oriented, functional, and multi-paradigm languages. Modern languages such as Python, Rust, Go, and TypeScript highlight other aspects, such as artificial intelligence (AI), memory safety, scalability, and developer productivity [1].

AI has emerged as a key player in programming language development over the past few years. Large-scale language models (LLMs) can create, explain, debug, translate, and document source code using natural language commands. According to a

*Author for Correspondence

Nidhi Chahal
E-mail: nidhi.ece@cgce.edu.in

¹Assistant Professor, Electronics and Communication Engineering, Chandigarh Engineering College-CGC, Landran, Mohali, Punjab, India

²⁻⁴Student, Electronics and Communication Engineering, Chandigarh Engineering College-CGC, Landran, Mohali, Punjab, India

Received Date: September 26, 2026
Accepted Date: September 28, 2026
Published Date: September 29, 2026

Citation: Nidhi Chahal, Purshotam Jha, Sonia, Vaishanvi. AI-Assisted Programming, Memory Safety, Multi-Paradigm Design and Emerging Development Models. Recent Trends in Programming Languages. 2026; 13(2): 43–50p.

2025 Stack Overflow Developer Survey, 84% of respondents are leveraging or intending to leverage AI tools when developing software, and 46% are not sure of the accuracy of AI-generated output. This means that the use of AI for programming is widespread, and reliable programming is still a major concern.

In addition to AI-assisted programming, there are other trends that are shaping the modern software development landscape, including memory-safe programming, multi-paradigm design, static typing, programming using natural language, and AI coding agents. The programming toolbox is not only changing, but so are the skills and duties of programmers.

This paper summarizes the major recent trends in programming languages, analyzes their advantages and problems, and suggests their future directions. Special focus is placed on AI-assisted programming, LLMs, memory safety, modern language ecosystems, and human–AI collaboration in software development.

DISCUSS THE EVOLUTION OF LANGUAGES: TALK ABOUT THE EVOLUTION OF PROGRAMMING LANGUAGES

Programming languages have progressed from low-level machine instructions to high-level and multi-paradigm languages that promote productivity, portability, safety, and maintainability. Early languages, such as Assembly and C, offered close control over hardware, whereas languages such as C++, Java, and C# promoted an object-oriented approach to the development of complex software systems.

As the Internet grew, programming became increasingly scripted, and languages such as JavaScript, Python, PHP, and Ruby became popular for the web. More recently, languages such as Rust, Go, Kotlin, and TypeScript have gained popularity for their various features, such as memory safety, concurrency, scalability, portability, and developer productivity.

One of the significant recent advancements is the incorporation of AI into programming. LLMs can also produce and edit source code based on natural language requirements, establishing a new level of abstraction between developers and programming languages. As a result, modern programming is shifting from manual code writing to AI-assisted code writing, automated testing, and human verification.

The fact that these developments are appearing demonstrates that hardware and software needs are no longer the only forces acting on the evolution of programming languages. Now, programming languages are designed and chosen based on other considerations, such as security, AI, developer experience, ecosystem support, and automated development.

RECENT TRENDS IN PROGRAMMING LANGUAGES

AI-Assisted Programming

AI has emerged as one of the biggest software development trends in recent times. LLMs can create, interpret, troubleshoot, translate, and document code with human-like text commands. AI coding assistants can also help developers write functions, create test cases, find bugs, and comprehend unfamiliar code [2].

In the 2025 Stack Overflow Developer Survey, 84% of respondents indicated that they were using or planning to use AI tools in their development process, highlighting the accelerated adoption of AI-powered programming. Despite these benefits, not everyone is convinced of the accuracy of AI-generated content, as 46% of respondents said that they did not trust the accuracy of AI output. This means that AI can be used to enhance productivity in development processes without the need for manual checks.

Thus, AI-assisted programming is a paradigm shift from manual programming to a cooperative programming approach.

The feedback from the tests is used to verify and have the AI code reviewed by a human.

The role of programming languages remains crucial, as AI-generated code must adhere to the syntax, semantics, type constraints, libraries, and runtime requirements of the target language. New environments will likely feature more sophisticated security, static analysis, and testing, along with AI assistance.

Large Language Models for Code Generation

LLMs offer a novel way of software development by enabling developers to create software from natural language descriptions. These models can be used to assist in code completion, program generation, debugging, code translation, documentation, and test case creation.

However, recent studies indicate that the use of LLMs is growing at various points in the software development lifecycle [3]. Hou et al. conducted a systematic literature review of 395 studies on the application of LLMs in software engineering to uncover major applications, reliability, evaluation, and practical challenges.

Even with their powers, LLM outputs are not necessarily correct. Logical errors, misuse of library functions, security weaknesses, and inefficient code implementation are possible errors that could be contained in the generated code. Therefore, human review, automated testing, and static analysis are still relevant.

As a result, the role of a programmer is shifting from being a coder to a requirement specifier, a test and quality tester, a coder, and a debugger. It is expected to shape the future of programming languages and development environments used, with a focus on improving the capabilities of these tools and allowing for more seamless integration with AI systems.

AI Coding Agents

AI coding agents take AI-assisted programming one step further by executing a series of related coding tasks. An agent could instead review a software repository, make plans for changes, edit files, run tests, find errors, and suggest additional changes.

A simplified process would be:

Requirements, Repository Analysis, Planning, Code Modification, Testing, Error Correction, and Human Approval.

AI agents can help minimize repetitive development tasks and automate parts of the software development lifecycle. However, giving autonomy more freedom brings with it more issues of reliability, security, privacy, access control, and unexpected changes.

According to the 2025 Stack Overflow Developer Survey data, AI agents are still in the nascent stages of development, and developers are deeply worried about the accuracy of AI agents and security [4].

Memory-Safe Programming

In contemporary programming language design, security and reliability have become significant criteria. While traditional systems languages offer a wealth of memory control, programming mistakes can result in vulnerabilities such as buffer overflows, use-after-free, and memory corruption.

One such development is Rust. The compiler can identify several memory safety issues during the compilation process. This is part of the trend toward secure-by-design programming languages, which have compilers and languages that catch some types of bugs, not just the programmer [5].

Security-sensitive applications, such as operating systems, embedded systems, browsers, and networking software, are especially important for memory-safe programming.

Python and AI-Oriented Language Ecosystems

Python is a crucial language in AI, machine learning, data science, automation, and scientific computing. It is relatively easy to use, and it has a large number of libraries and frameworks that have contributed to its wide adoption.

According to the 2025 Stack Overflow Developer Survey, Python's use has risen by seven percentage points since 2024, related to its growth in use for AI and data science and for supporting backend development.

This shows how crucial the ecosystem of a programming language is to its success, as well as its surrounding documentation, development tools, package repositories, frameworks, and community support.

Type-Safe Web Programming

JavaScript has been a traditionally important language in web development. TypeScript is an extension of JavaScript, with it being statically typed and supporting additional development time checks but still being compatible with the JavaScript ecosystem [6].

Certain types of errors can be caught even before the program is run, and static typing can aid program navigation, refactoring, documentation, and support for program development tools. Type-safe approaches are becoming more popular because of the general trend toward increased reliability and maintainability in large software projects.

Distribution of programming paradigms by typical application

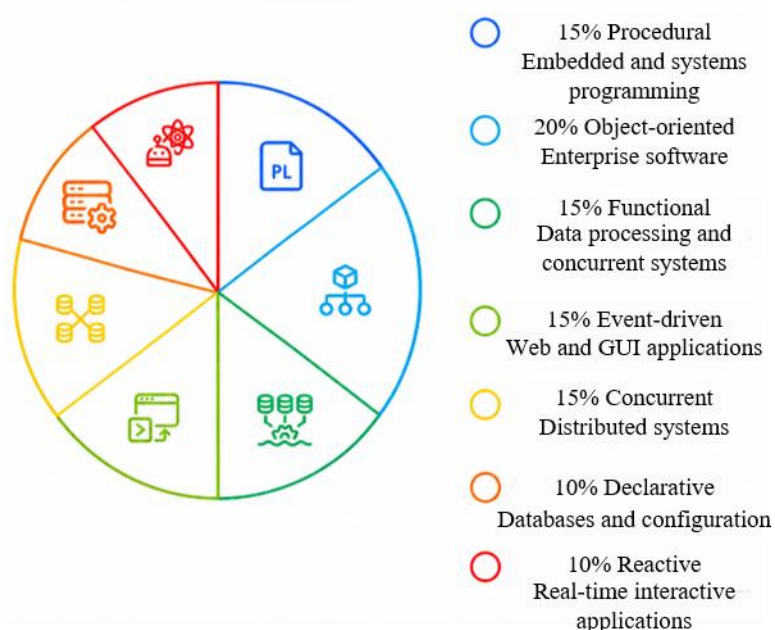


Figure 1. Multi-paradigm programming.

Multi-Paradigm Programming

Modern programming languages have multiple programming paradigms, as opposed to a specific programming model. There are a variety of programming paradigms, such as procedural, object-oriented, functional, concurrent, event-driven, and declarative, which can be used together in a single development environment (Figure 1).

Multi-paradigm programming provides flexibility and enables developers to choose the best method for any part of the software system. This is particularly helpful in today's applications, which include web interfaces, databases, cloud services, AI models, and distributed elements.

The program supports both natural language and low-code programming. The program is compatible with both natural language and low-code programming.

Another new trend is natural language programming and low-code/no-code platforms. For some applications, AI systems can convert user requirements translated into natural language into programming constructs, which can minimize the amount of manual code necessary to create a program.

A streamlined procedure is:

Natural Language Requirement → Program Design → Source Code → Testing → Human Validation

This can help make software development more accessible to users who may not have extensive programming experience. However, with the ambiguity of natural language and the need for algorithms, architecture, security, databases, and software testing in complex applications, natural language is insufficient [7].

Therefore, natural language programming is more aptly thought of as an extra layer of abstraction rather than a substitute for traditional programming.

Table 1. Recent trends analysis.

Trend	Representative technology	Primary focus	Major advantage	Key challenge
AI-assisted programming	LLM-based coding tools	Developer productivity	Faster code generation	Reliability of generated code
AI coding agents	Agentic development tools	Workflow automation	Automates multi-step tasks	Security and control
Memory-safe programming	Rust	Software security	Prevents many memory errors	Learning complexity
AI-oriented ecosystems	Python	AI and data science	Extensive libraries	Performance for some workloads
Type-safe web development	TypeScript	Reliability	Early error detection	Added type-system complexity
Multi-paradigm programming	Python, C++, Kotlin	Flexibility	Supports multiple approaches	Greater language complexity
Natural Language programming	LLM interfaces	Accessibility	Easier software creation	Ambiguous requirements
Low-code/no-code	Visual development platforms	Rapid development	Reduced manual coding	Limited customization

A RECENT TRENDS ANALYSIS FOR PROGRAMMING LANGUAGES

Analysis of recent trends in programming languages. Various methods to enhance software productivity, security, performance, and maintainability have been developed recently in the field of programming language development. AI-assisted programming is all about minimizing development time, while memory-safe languages are all about security and reliability. Likewise, multi-paradigm languages are flexible and type-safe languages that help detect errors during development (Table 1).

A comparison indicates that none of the trends discussed above is applicable to all the needs of modern software development. While memory-safe and type-safe languages emphasize reliability and security, AI-based approaches are more oriented towards productivity. Multi-paradigm languages offer flexibility, and natural language and low-code approaches minimize the technical aspects of software creation. Therefore, the next generation of programming environments will probably rely on a combination of these options, rather than just one technology.

A natural language requirement can then be used to create TypeScript and Rust code in an AI development environment, where it can then undergo automated testing or static analysis and be approved by a human before deployment. The synergy of this can offer productivity gains of AI and reliability assurance of modern programming languages.

THE DEVELOPMENT OF SOFTWARE HAS BEEN AFFECTED: SOFTWARE PRODUCTION HAS BEEN INFLUENCED

Programming languages have undergone immense changes in recent years owing to some trends. Some trends in programming languages are making significant changes in software development. By streamlining tasks such as code generation, documentation, and testing, AI-assisted tools can save time on repetitive tasks. Modern languages also contribute to the reliability of software in various ways, including static typing, memory safety, and compiler features for detecting errors.

Although AI can enhance productivity, it is essential to have the code tested and reviewed by humans to ensure that it is accurate and secure, as it can still generate incorrect or insecure code. Other programming errors and security vulnerabilities can be minimized in memory-safe and type-safe languages [8].

These changes are also affecting the role of developers. The focus of programmers is shifting from writing source code to learning how to use AI tools, as well as other skills relevant to software systems, such as system design, code evaluation, testing, security, and debugging.

In summary, the future of modern programming is a blend of automation, security, productivity, and collaboration between humans and AI.

CHALLENGES AND LIMITATIONS

While the benefits of using these programming trends are significant, there are some obstacles to overcome. Human verification is required for AI-generated code, as it can have logical flaws, security issues, or suboptimal implementations. Issues related to privacy and intellectual property can also arise in the use of source code by external AI systems.

Overdependence on AI is another challenge, as it can lead to a lack of understanding of basic programming principles among developers. Modern languages may also possess complicated characteristics that add to the difficulty of learning the language for beginners.

The added concerns with AI coding agents are that access, monitoring, testing, and approval systems need to be put in place to support increased automation.

Thus, any future programming environment will need to be a compromise between automation and reliability, security, transparency, and human control.

FUTURE DIRECTIONS

Future programming environments will probably be more closely tied to AI and programming languages. AI-native development environments can generate code while also conducting testing, debugging, security analysis, and documentation.

Another significant option is verified code generation, in which AI-generated code is automatically verified through static analysis, testing, and formal verification methods. It is also possible that memory-safe and type-safe language features might be gaining greater significance for secure software development [9].

While AI coding agents can assist with more complex development processes, human oversight will continue to be crucial for requirements, architecture, security, and approvals. Software development for specialized applications may become even easier with the use of natural language programming and domain-specific languages.

The future will see increased focus on cooperation between humans and AI, languages that are built with security in mind, automated verification, and intelligent programming environments, but not on the total replacement of traditional programming.

RESEARCH GAPS

Even though programming languages and AI-assisted development have come a long way, there are still important areas that need more study. One key issue is finding ways to check if code written by AI is correct and safe. Another challenge is creating methods to turn simple language instructions into exact programming plans.

More work is also needed on AI-native programming languages, building code that can be verified for correctness, designing secure AI coding agents, improving how humans and AI work together, and changing the way we teach programming. Creating benchmarks would make it easier to compare AI-generated programs across different programming languages and tasks [10].

Solving these gaps can lead to programming environments that provide users with the speed and convenience of AI tools while maintaining the reliability of traditional programming and verification methods.

CONCLUSION

Programming languages are constantly changing to keep up with the needs of modern software development. New ideas such as programming, LLMs, AI coding agents, memory-safe languages, type-safe development, and multi-paradigm programming are having a significant impact on how we build software.

AI has opened up ways to generate code and automate tasks, and the latest language features have made software safer, more reliable, and easier to maintain. However, problems with correctness, security, privacy, and heavy reliance on AI remain serious concerns.

I believe that in the future, programming will probably center on human–AI collaboration rather than full automation. By pairing development tools with secure programming languages, automated testing, and human verification, we can create software development environments that are more productive and trustworthy.

REFERENCES

1. Stack Overflow. (2025). AI: 2025 Stack Overflow Developer Survey. [online] Stack Overflow. Available from: <https://survey.stackoverflow.co/2025/ai>.
2. Hou X, Zhao Y, Liu Y, Yang Z, Wang K, Li L, et al. Large language models for software engineering: a systematic literature review. *ACM Trans Softw Eng Methodol*. 2024;33(8):1-79. doi:10.1145/3695988.
3. Jiang J, Wang F, Shen J, Kim S, Kim S. A survey on large language models for code generation. *ACM Trans Softw Eng Methodol*. 2026;35(2):1-72. doi:10.1145/3747588.
4. He J, Treude C, Lo D. LLM-based multi-agent systems for software engineering: literature review, vision, and the road ahead. *ACM Trans Softw Eng Methodol*. 2025;34(5):1-30.
5. Rust Project Developers. (2025). The Rust programming language [Online]. Rust Programming Language Documentation. Available from: <https://doc.rust-lang.org/book/>.
6. Microsoft. (2025). TypeScript handbook [Online]. TypeScript Documentation. Available from: <https://www.typescriptlang.org/docs/handbook/>.
7. Fan DP, Ji GP, Sun G, Cheng MM, Shen J, Shao L. Camouflaged object detection. In: 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). 2020. p. 2774-2784. doi:10.1109/CVPR42600.2020.00285.
8. Sun Y, Chen G, Zhou T, Zhang Y, Liu N. Context-aware cross-level fusion network for camouflaged object detection. In: Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence. International Joint Conferences on Artificial Intelligence Organization; 2021. p. 1025-1031. doi:10.24963/ijcai.2021/142.
9. Chen G, Liu SJ, Sun YJ, Ji GP, Wu YF, Zhou T. Camouflaged object detection via context-aware cross-level fusion. *IEEE Trans Circuits Syst Video Technol*. 2022;32(10):6981-6993. doi:10.1109/TCSVT.2022.3178173.
10. Zhang S, Kong D, Xing Y, Lu Y, Ran L, Liang G, et al. Frequency-guided spatial adaptation for camouflaged object detection. *IEEE Trans Multimed*. 2025;27:72-83. doi:10.1109/TMM.2024.3521681.