Published by :
https://www.ijert.org/
An International Peer-Reviewed Journal

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 03 , March - 2026

CodeSage: An AI-Enhanced IDE with Visual Execution Flow for Novice Programmers

M. Nageswara Rao, R. Prasanna Kumar, K. Naga Manjusha, M. Mohith, A. Venkata Ramana, S. Navya Sri
Sri Vasavi Engineering College,Tadepalligudem

Abstract - In today's rapidly evolving software development landscape, tools for novice programmers must do more than simply allow code writing; they should guide learners, simplify complex processes, and improve productivity. CodeSage is an interactive development environment designed to function both as a coding tool and a learning assistant. It combines real-time syntax- aware code editing, compiler support, and a visual execution tracer that helps users understand how programs run. Unlike traditional IDEs that only compile and run code, CodeSage focuses on improving conceptual understanding by visualizing program flow, memory usage, and the step-by-step execution of instructions. Built on the powerful Monaco Editor, the system provides a modern coding interface connected to a secure backend interpreter. Additionally, an optional AI-powered explanation feature helps learners understand programming concepts more clearly by explaining code in simple terms. This paper presents the design, architecture, and educational purpose of CodeSage, a tool aimed at making programming more interactive and easier to understand for beginners, educators, and self-learners.

Keywords: Code Editor, Compiler Integration, Visual Execution, AI-Powered Code Assistant, Syntax Analysis,Code Visualization, Debugging Tools.

INTRODUCTION

The Persistent Challenge of Introductory Programming

Learning to program is widely considered difficult for many students. Research in computer science education has consistently reported high failure and dropout rates in introductory programming courses (CS1). Studies also show that student engagement in computing courses is often lower than in many other academic fields.

These difficulties arise from several factors. Beginners must learn abstract programming logic, strict language syntax, and the use of development tools simultaneously. Concepts such as variables, loops, and conditional statements often become obstacles. Many novice programmers also struggle to understand what happens internally when a program runs. Although they may understand individual statements, they find it difficult to see how those statements interact to form a working program. This gap between written code and program execution highlights the need for better educational tools to support learning.

The Gap in Existing Tooling

The programming environment plays an important role in learning. While professional IDEs such as Visual Studio Code, IntelliJ IDEA, and Eclipse offer powerful features, their complex interfaces can overwhelm beginners and increase cognitive load. Pedagogical programming environments simplify interfaces to support learning, but they often lack flexibility for advanced tasks. CodeSage bridges this gap by combining a professional editor with a simplified, learning-focused environment.

The Emergence of AI in Education

Recent advancements in **Large Language Models (LLMs)** such as ChatGPT, Codex, and Gemini have influenced programming education by providing tools that can generate, explain, and debug code. These AI systems are increasingly used in introductory programming courses to offer guidance and support learning. However, over-reliance on AI-generated solutions may weaken students' understanding and problem-solving skills. Issues such as inaccurate responses, AI "hallucinations," and concerns about academic integrity highlight the need for careful integration of AI tools in programming education.

Introducing CodeSage and Our Contribution

This paper introduces **CodeSage**, an interactive development environment designed to support beginner programmers by combining features from professional IDEs, program

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 03 , March - 2026

visualization tools, and AI- assisted learning. The system integrates the **Monaco Editor**, a step-by-step execution visualizer, and an AI assistant that explains code rather than generating solutions. By bringing these features into a single platform, CodeSage aims to improve understanding of programming concepts and support programming education.

Related Work

Pedagogical Development Environments vs. Professional IDEs

The choice of a programming environment for beginners has long been debated in computer science education. While professional IDEs offer powerful features such as debugging tools and version control, their complex interfaces can overwhelm novice programmers and increase cognitive load. To address this, pedagogical development environments (PDEs) like BlueJ, Code::Blocks, and IDLE provide simplified interfaces and learning-focused features. However, these tools often lack the flexibility required for advanced programming tasks. CodeSage addresses this issue by combining the professional **Monaco Editor** with a simplified interface designed to support beginner learning.

The Role of Program andAlgorithm Visualization

A significant body of research in computer science education supports the use of visualization to make abstract computational concepts more concrete and understandable for learners.[18] By providing a visual representation of a program's execution or an algorithm's operation, these tools help students build accurate mental models, reduce cognitive load, and engage in active experimentation rather than passive study.[18] Visualizations can range from high-level animations of sorting algorithms to low-level depictions of runtime state, including stack frames, heap objects, and variable bindings.[18]

Limitations of Standalone Visualizers

Standalone visualization tools like **Python Tutor** are useful for learning but have several limitations. They are mainly designed for demonstration and usually support only small code examples, lacking features needed for complex programs such as external libraries or file handling. Another limitation is that students must copy code from their IDE into the visualizer, which disrupts the workflow. CodeSage addresses this issue by integrating visualization directly into the code editor, allowing students to write, execute, and understand code within a single environment.

Large Language Models in Programming Education

The integration of LLMs into educational settings is a recent but rapidly expanding field of research. Systematic literature reviews analyzing studies published between 2022 and 2025 reveal a surge of interest in applying models like ChatGPT and GitHub Copilot to programming instruction, predominantly at the university level.[10] This research explores a spectrum of applications, from using LLMs as learning assistants to evaluating their performance on academic programming
tasks.[11]

Benefits and Opportunities

Large Language Models (LLMs) provide several benefits in programming education. They can act as virtual tutors by offering instant feedback and personalized explanations, helping students improve engagement and performance in coding tasks. These tools can also automate repetitive activities such as generating boilerplate code or comments, allowing learners to focus on problem-solving and algorithm design. Additionally, instructors can use LLMs to create learning materials and answer common questions, while students gain familiarity with AI- assisted coding used in modern software development.

Challenges and Risks

Despite their advantages, the use of Large Language Models (LLMs) in education raises several concerns. A key issue is student over-reliance on AI-generated solutions, which may lead to a superficial understanding of programming concepts and weaker problem-solving skills. LLMs can also produce incorrect or inefficient code, known as “hallucinations,” which beginners may struggle to evaluate. In addition, AI tools capable of solving assignments raise concerns about academic integrity. These issues highlight the importance of carefully integrating AI tools into the learning process.

THE CODESAGE SYSTEM:

Architecture and Features

High-Level System Architecture

CodeSage is designed as a modern web application using a **client–server architecture**. This approach allows users to access the system through a web browser without requiring local installation, while computational and security-sensitive tasks are handled on a dedicated backend server.
The system consists of two main components.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 03 , March - 2026

1. **Frontend Client:** A single-page application built with the React framework that provides the user interface, including the code editor, visualization panel, and output console. It manages user interactions and communicates with the backend through RESTful APIs.
2. **Backend Service:** A server application developed using Spring Boot that handles request processing, secure execution of user-submitted code in isolated environments, integration with the AI model for code explanations, and returning outputs such as program results, errors, and execution traces.

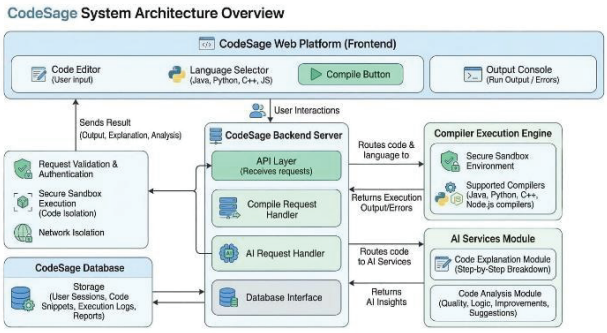

Fig.1: Architecture

Frontend: The Interactive Learning Environment

Monaco Editor Integration

The core component of CodeSage is the **Monaco Editor**, the open-source editor used in Visual Studio Code, which provides a familiar and professional coding environment for learners. It is integrated into the React frontend using **@monaco-editor/react**, enabling features such as syntax highlighting, IntelliSense with auto-completion, and real-time error detection. These features help beginners write correct code more easily and focus on programming logic and problem-solving.

Backend: Secure Code Execution and Orchestration

The backend of CodeSage is responsible for the critical tasks of executing user code securely and interfacing with the AI service. The choice of technologies for the backend reflects a focus on robustness, security, and ease of development.

Spring Boot API Service

The backend of CodeSage is implemented as a RESTful API service using the Java-based Spring Boot framework. Spring Boot was chosen because it enables the development of stand-alone applications with minimal configuration. It also includes an embedded web server such as Tomcat, which simplifies deployment and provides strong support for building web APIs. The Spring Boot service acts as the orchestration layer of the system. It exposes API endpoints like /execute and /explain that are used by the React frontend. When code is submitted to the /execute endpoint, the backend runs the code in a sandboxed environment and collects the results, including program output, errors, and execution traces. These results are then returned to the frontend in JSON format, ensuring secure separation between application logic and code execution.

Sandboxed Execution with Docker

Executing user-submitted code poses significant security risks, as it may contain malicious commands or errors that could harm the server. To address this, CodeSage uses **Docker-based sandboxing**, ensuring that all code runs in a temporary and isolated environment without access to the host system or network. This approach follows best practices for executing untrusted code. The combination of **Spring Boot for orchestration** and **Docker for isolation** helps maintain a secure and reliable execution environment.

The execution workflow managed by the Spring Boot service is as follows:

1. **Receive Request:** The backend receives a POST request at the /execute endpoint containing the source code and selected language.
2. **Prepare Execution:** The code is stored temporarily in a secure directory on the server.
3. **Start Container:** The backend uses Java's ProcessBuilder to run a new Docker container (e.g., docker run --rm).
4. **Configure Sandbox:** A minimal language-specific image (such as python:3.11-slim or openjdk:17-slim) is used. The code file is mounted as read-only, networking is disabled, and the container runs with limited permissions.
5. **Execute Code:** The interpreter or compiler runs inside the container along with a tracing tool that records variable states and call stack information.
6. **Capture Output:** Program output, errors, and execution trace

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 03 , March - 2026

data are collected from the container.

7. **Destroy Container:** The container is automatically removed after execution to ensure no data persists between runs.

8. **Return Response:** The results are packaged into a JSON response and sent back to the frontend.

AI Integration with Gemini API

The **AI Sage** feature in CodeSage uses the **Google Gemini API** to provide code explanations. When a user selects a code snippet, the request is sent to the **/explain** endpoint on the Spring Boot backend, which securely forwards it to the Gemini API. The AI then generates a beginner-friendly explanation that is returned to the frontend and displayed to the user. This feature provides immediate, contextual support within the coding environment while keeping the API key secure.

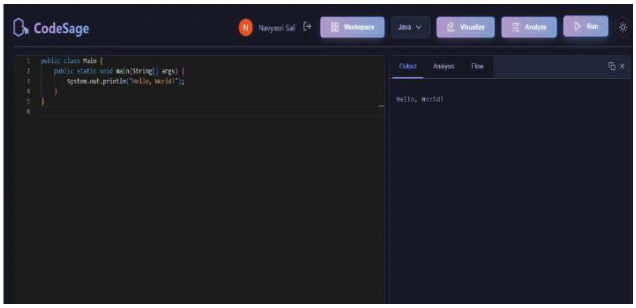

Fig.2: Program Output

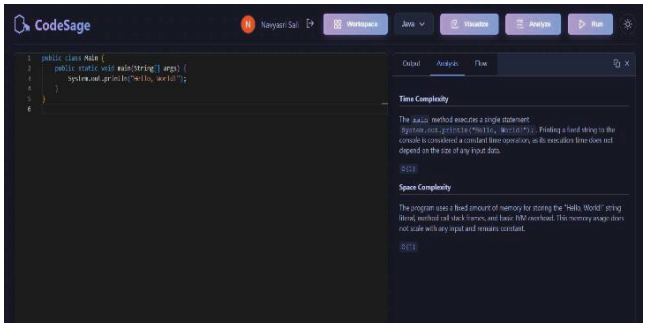

Fig.3: AI Code Analysis

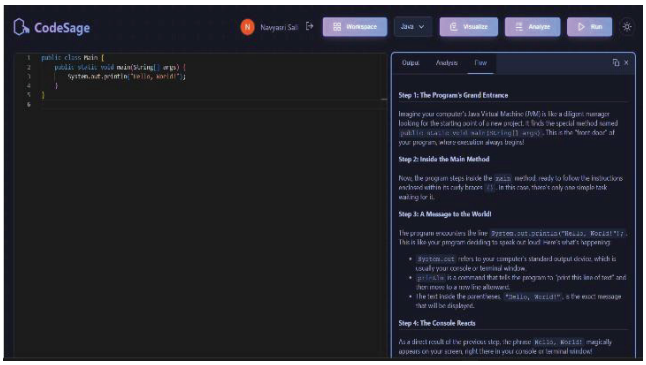

Fig.4: AI Code Visulaization

Pedagogical Implications and Discussion

Reducing Cognitive Load and Fostering Mental Models

A major challenge for novice programmers is developing a clear mental model of the "notional machine," the system that executes their code. Without this understanding, it is difficult to relate written code to its runtime behavior. CodeSage addresses this through its visualization feature, which displays the call stack, memory allocation, and variable changes step by step, helping students understand how programs execute. It also reduces cognitive load by using the professional Monaco Editor with a simplified interface that highlights only essential features, allowing beginners to focus on programming logic rather than navigating complex tools.

A Comparative Analysis

To situate CodeSage within the existing ecosystem of programming tools for novices, it is useful to compare its features and pedagogical approach against the two dominant categories: professional IDEs (often augmented with AI code generators) and standalone program

visualizers. The following table provides a comparative analysis across several key dimensions.

Feature/Attribute	Professionl IDE(e.g.,VS Code)	Standalone Visualizer (e.g.Python Tutor)	CodeSage
Primary Goal	Professionl Productivity	Conceptual Demonstration	Integrated Learning & Development

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 03 , March - 2026

Interface Complexity	High (Many panels, menus, extensions)	Very Low (Code and visualizatin only)	Medium (Curated profession al interface)
Scope of Use	Full-scale software develo pment	Small, self-contained code snippets	Small to medium sized education al projects
Execution Visualization	Via complex debugger (break	Core feature, explicit and automatic	Core feature, integrated into ID E workflow
AI Assistance Model	Code Generation (e.g., Copilot)	N/A (or limited AI help)	On-Demand Code Explanation
Pedagogical Focus	None (Focus on efficie ncy)	Unders tanding program state	Bridging theory (visualization) and practice (coding)

CONCLUSION AND FUTURE WORK

Summary of Contributions

This paper presented CodeSage, an AI-enhanced development environment designed to support novice programmers. It combines a professional code editor with learning-focused tools, including a step-by-step program visualizer that helps students understand program execution and build accurate mental models. CodeSage also provides an optional AI assistant for contextual code explanations and uses a Spring Boot backend with Docker-based sandboxing for secure execution. By integrating visualization, AI explanations, and a professional interface, CodeSage bridges the gap between simple educational tools and complex IDEs, helping learners develop both conceptual understanding and practical coding skills.

Limitations

As a prototype, CodeSage has several limitations. It currently supports only a limited number of programming languages, mainly Python, and the visualizer may not fully represent advanced language features or complex object-oriented structures. The AI integration relies on general prompting with the Gemini API and could be improved with more context-aware prompt design. Additionally, although CodeSage is based on established educational principles, its effectiveness has not yet been validated through formal user studies.

Future Work and Proposed Evaluation

The future development of CodeSage will proceed along two parallel tracks i.e system enhancement and rigorous empirical evaluation.

Works cites

1. CodeSage_Abstract.docx
2. Full article: Practical thinking while learning to program – novices' experiences and hands-on encounters, ccessed on September 24, 2025, https://www.tandfonline.com/doi/full/10.1080/08993408.2021.1953295
September 24, 2025, https://hrmars.com/papers_submitted/23238/programming-challenges-experience-by-primary-school-students-a-systematic-literaturereview.pdf
4. Strategies of Novice Programmers - CBS Research Portal, accessed on September 24, 2025, https://research.cbs.dk/files/58519067/Begum_N_rbjerg_Clemmensen_IRIS2018.pdf
5. A study of the difficulties of novice programmers - ResearchGate, accessed on September 24, 2025, https://www.researchgate.net/publication/220808194_A_study_of_the_difficulties_of_novice_program mers
6. Best IDE for Beginners: Simplifying the Coding Journey, accessed on September 24, 2025, https://www.modernagecoders.com/blog/best-ide-for-beginners
7. 19 Best IDE Software Picks of 2025 - The CTO Club, accessed on September 24, 2025, https://thectoclub.com/tools/bestide-software/
8. Comparing Popular IDE Tools for Developers - Dev Hunt, accessed on September 24, 2025, https://devhunt.org/blog/comparing- popular-ide-tools-for-

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 03 , March - 2026

develope
rs
9. (PDF) The effects of Professional and Pedagogical Program ..., accessed on September 24, 2025, https://www.researchgate.net/publ ication/220803521_The_effects_of _Professional_and_Pedagogical_Pr ogram_Development_Environmen ts_on_Novice_Programmer_Perce ptions
10. (PDF) Systematic Review of Large Language Model Applications in Programming Education, accessed on September 24, 2025, https://www.researchgate.net/publ ication/395563325_Systematic_Re view_of_Large_Language_Model_ Applications_in_Programming_Edu cation

REFERENCES

[1] A. Ahmad, M. Khan, and R. Gupta, "AI-Powered Code Understanding: A Survey on Code Summarization and Explanation Techniques," IEEE Access, vol. 9, pp. 112345–112359, 2021.

[2] T. Chen, Y. Liu, and M. Sun, "CodeBERT: A Pre-Trained Model for Programming and Natural Languages," in Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP), 2020, pp. 1536–1547.

[3] M. Allamanis, E. T. Barr, P. Devanbu, and C. Sutton, "A Survey of Machine Learning for Big Code and Naturalness," ACM Computing Surveys, vol. 51, no. 4, pp. 1–37, 2018.

[4] J. R. Cordy, "Comprehending Programs through Dynamic Visualization," in IEEE International Workshop on Program Comprehension, 2019, pp. 12–19.

[5] A. Vaswani et al., "Attention Is All You Need," in Advances in Neural Information Processing Systems (NeurIPS), 2017, pp. 5998–6008.

[6] K. Li, S. Feng, and J. Huang, "AI-Assisted Code Review and Bug Detection using Transformer Models," IEEE Transactions on Software Engineering, vol. 48, no. 11, pp. 4210–4222, 2022.

[7] S. Ray and N. Kumar, "Interactive Visual Debugging Tools for Learning Programming Concepts," in Proceedings of the IEEE Conference on Learning and Teaching in Computing and Engineering (LaTiCE), 2020, pp. 45–52.

[8] P. Jain and R. Singh, "Integrating Compiler Visualization and AI Feedback for Programming Education," International Journal of Educational Technology in Higher Education, vol. 19, no. 5, 2023.

[9] D. Li and K. S. Chan, "Real-Time Code Execution Analysis and Visualization Framework for Teaching Programming," in IEEE Global Engineering Education Conference (EDUCON), 2021, pp. 1340–1348.

[10] M. Chen et al., "Evaluating Large Language Models Trained on Code," arXiv preprint arXiv:2107.03374, 2021.