JAVAAI COACH: A VOICE-BASED GENERATIVE AI TUTOR FOR JAVA LEARNERS

Shaikh Firoz Shaikh Ismail, Vinit V. Sarnaik, Shreyas D. Sakhare
[1]Computer Science and Engineering, Babasaheb naik college of engineering, Pusad-445204, Maharashtra, India
firoz.s.ismail@gmail.com
[2]Computer Science and Engineering, Babasaheb naik college of engineering, Pusad-445204, Maharashtra, India
vinitsarnaik2502@gmail.com
[3]Computer Science and Engineering, Babasaheb naik college of engineering, Pusad-445204, Maharashtra, India
shreyashsakhare938@gmail.com

ABSTRACT
This paper introduces an AI-powered chatbot, developed using advanced data science and machine learning methods, to support the teaching of Java programming. It is designed for today's educational needs, offering interactive assistance to help students debug code, understand core Java concepts, and solve programming problems more independently. In the modern world, where personalized digital education is in high demand, such tools can make high-quality programming instruction more accessible even in environments with limited teacher support. The adoption of this system can elevate student confidence, reduce frustration with errors, and narrow learning gaps, potentially transforming how Java and other programming languages are taught in classrooms and online courses. This JavaAI Coach, a voice-based intelligent tutoring system designed to support novice programmers in learning Java. Leveraging automatic speech recognition (ASR), large language models (LLMs), and text-to-speech (TTS), the system provides real-time spoken feedback, proactive hints, and Socratic-style guidance. We position JavaAI Coach within the broader landscape of intelligent tutoring systems (ITS) and proactive programming assistants, comparing it with recent works in the domain. A detailed system architecture, development roadmap, and evaluation framework are proposed. The contributions lie in accessibility, context-awareness, and proactive voice interaction, which together aim to reduce cognitive load and enhance programming education outcomes.
Keywords: *Python Programming, Intelligent Tutoring Systems, Code Debugging, Ethical AI, AI in Education*

I. INTRODUCTION

Programming literacy is increasingly regarded as a core skill in today's technology-driven society. Among the many programming languages available, Java occupies a unique and enduring role. It powers enterprise back-end systems, Android applications, large-scale distributed platforms, and embedded systems. Its longevity and widespread adoption in universities make Java a cornerstone in computer science curricula worldwide. For learners, mastering Java offers not only career opportunities but also exposure to rigorous programming concepts such as object-oriented design, type safety, and memory management. However, the language's strict syntax and extensive libraries also present a steep learning curve for beginners, who often face frustration and disengagement when encountering compiler errors or debugging challenges.

At the same time, artificial intelligence (AI) is reshaping education. Advances in large language models (LLMs), speech technologies, and adaptive learning platforms make it possible to create systems that can personalize feedback, simulate one-on-one tutoring, and provide continuous support outside of formal classroom hours. Studies in AI-powered intelligent tutoring systems (ITS) show that, when designed effectively, they can rival or even outperform traditional teaching interventions in certain contexts. For learners, AI tutors offer immediacy and scalability—qualities particularly valuable in programming education, where class sizes are often large and access to individualized mentoring is limited.

Despite this promise, most existing AI tutors for programming rely heavily on text-based interaction. Learners must still type questions into chatbots or read lengthy textual explanations, which can be cognitively taxing when combined with the already challenging task of writing and debugging code. A voice-based interface offers a compelling alternative. By engaging learners in spoken dialogue, tutoring can feel more natural and less intrusive, reducing cognitive load and enhancing accessibility for learners with visual impairments or limited typing proficiency. Furthermore, the act of verbalizing problems and reasoning aloud has been shown to reinforce learning and metacognitive awareness.

In this context, the integration of Java education with conversational AI has both technical and pedagogical significance. Technically, it demonstrates the feasibility of combining automatic speech recognition (ASR), natural language generation (NLG), and text-

to-speech (TTS) within a latency-sensitive loop tailored for programming tasks. Pedagogically, it explores new modalities of support—such as Socratic questioning, proactive hinting, and contextual feedback—that align with educational best practices. JavaAI Coach is positioned at the intersection of these developments, seeking to make programming education more interactive, inclusive, and effective for today's learners.

II. LITERATURE REVIEW

The integration of artificial intelligence (AI) into programming education has opened new avenues for personalized and scalable learning support. Traditional programming instruction often relies on static materials, limited feedback loops, and overburdened instructors, leaving many novice learners struggling with syntax errors, debugging, and conceptual misunderstandings. AI-driven tutoring systems attempt to bridge this gap by providing immediate, adaptive, and context-sensitive assistance. This section reviews prior work on intelligent tutoring systems (ITS), large language models (LLMs), and multimodal educational interfaces, highlighting the progress and limitations that inform the design of JavaAI Coach.

A. From Intelligent Tutoring Systems to AI Tutors

The roots of ITS date back to the 1980s, when rule-based systems such as Carnegie Mellon's Cognitive Tutor demonstrated that adaptive guidance could improve test scores by dynamically tailoring problem difficulty. Later systems, such as AutoTutor, expanded this approach with natural language dialogues, allowing learners to engage in conversational problem-solving. Meta-analyses confirm that ITS consistently produce learning gains across K–12 and higher education settings, though concerns remain regarding scalability, methodological rigor, and domain transferability (Létourneau et al., 2025).

B. Rise of Large Language Models in Programming Education

With advances in natural language processing (NLP) and machine learning, programming tutors evolved from rigid, rule-based systems to data-driven, generative ones. Models such as Code Llama and WizardCoder are specifically optimized for code generation and explanation, achieving strong performance on benchmark tasks (Rozière et al., 2023; Luo et al., 2023). More specialized systems like LPITutor integrate fine-tuned pedagogical strategies, offering clearer guidance and improved learner comprehension (Liu et al., 2025). Empirical studies (Mueller & List, 2025; Zhao et al., 2025) show that LLM-based tutors can provide context-rich, proactive hints directly within development environments, reducing trial-and-error cycles and improving problem-solving efficiency. However, such systems often prioritize correctness of output over the pedagogy of learning, leading to risks of over-reliance or shallow comprehension.

C. Proactive and Multimodal Tutoring

Recent research highlights the importance of moving beyond reactive text-based systems. Proactive tutors, such as CodingGenie, anticipate learner difficulties and deliver just-in-time hints (Zhao et al., 2025). Others, like TRAVER, incorporate trace-and-verify workflows to track student reasoning over time (Wang et al., 2025). Importantly, randomized controlled trials suggest that AI tutoring can outperform even in-class active learning when designed with pedagogical scaffolding (D'Andrea et al., 2025). Yet most of these systems rely heavily on text interfaces, which can increase cognitive load, particularly for novice programmers already managing complex debugging tasks.

A growing body of work advocates for multimodal learning support. Spoken interfaces, combining automatic speech recognition (ASR) and text-to-speech (TTS), offer a more natural, conversational modality. Research in cognitive psychology suggests that verbalizing reasoning enhances metacognitive awareness and retention. Despite this potential, voice-based tutoring for programming remains underexplored.

D. Identified Gaps

The literature establishes that ITS and LLM-based tutors significantly improve learning outcomes, but three gaps remain evident. First, text-centric interaction limits accessibility and may burden cognitive resources during programming. Second, most tutors emphasize solution delivery over conceptual scaffolding, leaving learners at risk of passive dependence. Third, proactive and multimodal designs are still emerging, with limited evaluation in programming contexts. Addressing these gaps motivates the development of JavaAI Coach, which integrates voice-based dialogue, context-aware feedback, and Socratic questioning to support novice Java learners in a more interactive and pedagogically grounded manner.

III. PROBLEM STATEMENT AND OBJECTIVES

The current ecosystem of programming education lacks an accessible, real-time, and context-aware tutoring system that can proactively guide learners through both syntactic and conceptual difficulties in Java. And they are text-centric. This gap highlights the need for a multimodal, voice-enabled, and pedagogically grounded solution.

Objectives:

1. Provide real-time spoken guidance on syntax and conceptual errors.
2. Integrate IDE context (code history, compilation errors) with the LLM reasoning pipeline.

3. Provide proactive, Socratic dialogue rather than purely reactive assistance.
4. Achieve low-latency, multimodal performance suitable for classroom deployment.

IV. COMPARISON

Table 1 summarizes comparisons between JavaAI Coach and closely related systems across modality, context integration, proactivity, pedagogy, and evaluation.

Work	Modality	Context integration	Proactivity	Pedagogy	Evaluation
JavaAI Coach (ours)	Voice + IDE	Strong (code + errors + speech)	Yes	Socratic, scaffolding	Planned RCT
Mueller & List (2025)	IDE text	Strong	Yes	Proactive hints	User study
Zhao et al. (2025)	IDE assistant	Strong	Yes	Customizable proactivity	Case analyses
Wang et al. (2025)	Agent workflow	Moderate	Yes	Knowledge tracing + verifier	Benchmarks
Liu et al. (2025)	Chat tutor	Moderate	Partial	Fine-tuned pedagogical agent	Rubric evaluation
Létourneau et al. (2025)	Meta-analysis	N/A	N/A	Varied	28 studies
D'Andrea et al. (2025)	AI tutor	Varies	Varies	Aligned pedagogical practices	Randomized trial

Table 1. Comparison of JavaAI Coach with related works

V. SYSTEM ARCHITECTURE

The system has five principal modules: (1) ASR for speech-to-text, (2) Context collector for IDE state and error traces, (3) LLM inference engine, (4) Dialogue manager for proactive timing and Socratic questioning, and (5) TTS for spoken feedback. The architecture emphasizes modularity, safety (sandboxed code execution), and a low-latency pipeline suitable for classroom use.

A. Automatic Speech Recognition (ASR)

The ASR module transcribes learner queries in real time. For accessibility and cost considerations, open-source systems such as Whisper or Vosk may be deployed locally, while cloud APIs can be integrated for higher accuracy. The ASR output is passed downstream with confidence scores, enabling filtering of uncertain transcriptions.

B. Context Collector

This module captures the learner's current IDE state, including code files, compilation logs, and recent error traces. Contextual integration ensures that feedback is grounded in the learner's actual work rather than generic responses. By combining spoken input with code context, the system provides precise, situationally relevant hints.

C. LLM Inference Engine

The LLM serves as the central reasoning unit. It processes ASR transcripts and contextual inputs, then generates pedagogically aligned outputs. Unlike generic code assistants, the LLM is configured to follow Socratic and scaffolding strategies rather than giving direct solutions. Safety filters also ensure that responses remain age-appropriate and pedagogically sound.

D. Dialogue Manager

The dialogue manager determines when and how to intervene. Instead of waiting for learners to ask for help, it can proactively suggest hints when repeated errors occur or when the learner appears stuck. It manages multi-turn interactions, maintains conversational state, and schedules interventions to avoid cognitive overload.

E. Text-to-Speech (TTS) Module

Finally, the TTS module converts generated feedback into natural spoken dialogue. This ensures that learners can stay focused on their code editor while receiving auditory guidance. Modern neural TTS engines (e.g., Tacotron, VALL-E) can provide expressive, human-like speech that enhances engagement.

F. Workflow Integration

Together, these modules form a closed feedback loop: learner input is captured as speech, contextualized with IDE data, processed through the LLM, managed by the dialogue engine, and delivered as voice feedback. The architecture is designed for modularity, allowing each component to be swapped or upgraded independently.

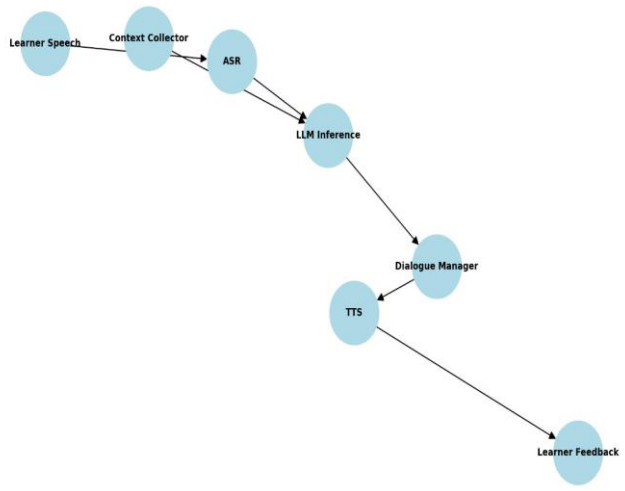

Fig. 1. Module Diagram.

VI. DEVELOPMENT ROADMAP

Phase 1: Prototype with open-source ASR and TTS; local LLM for proof-of-concept.
Phase 2: Integrate IDE plugin to capture code edits, run logs, and stack traces.
Phase 3: Implement proactive hint scheduler and Socratic dialogue policies.
Phase 4: Small-scale user studies for usability and iterative improvement.
Phase 5: Large-scale randomized controlled trial to measure learning gains.

VII. EVALUATION METRICS

Primary metrics: Answer accuracy (target ≥80%), Hint helpfulness (target ≥75%), ASR WER (target ≤12%), Response latency (target ≤2s), and User satisfaction (target ≥80%). Evaluation combines task-based performance, standardized rubrics, and subjective surveys.

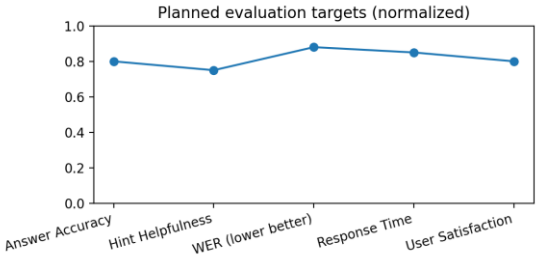

Fig. 2. Normalized target thresholds for primary evaluation metrics.

VIII. CONCLUSION

This study demonstrates the transformative potential of AI-powered tools in programming education, offering a scalable and accessible solution to long-standing pedagogical challenges. By integrating adaptive feedback, ethical safe- guards, and pedagogical best practices, the chatbot bridges the gap between theoretical instruction and practical coding proficiency. This concluding section summarizes the key achievements, contextualizes the work within broader educational frameworks, and outlines future directions for research and development.

JavaAI Coach demonstrates a design that merges voice interaction, context-aware LLM reasoning, and proactive pedagogy to address gaps in programming education. The work prioritizes accessibility and low-latency interactions, and proposes a rigorous evaluation plan. Future directions include expanded language support, analytics for personalized learning paths, and further safety evaluations.

REFERENCES

[1] Létourneau, A., Martineau, M. D., Charland, P., Karran, J. A., Boasen, J., & Léger, P. M. (2025). A systematic review of AI-driven ITS in K–12 education. NPJ Science of Learning, 10, 29. https://doi.org/10.1038/s41539-025-00320-7

[2] Mueller, M., & List, C. (2025). The power of context: An LLM-based programming tutor. ECSEE 2025. https://doi.org/10.1145/3723010.3723034

[3] Zhao, S., Zhu, A., Mozannar, H., Sontag, D., Talwalkar, A., & Chen, V. (2025). CodingGenie: A proactive programming assistant. arXiv. https://arxiv.org/abs/2503.14724

[4] Wang, Y., Zhang, H., Li, X., & Zhou, Y. (2025). Trace-and-Verify (TRAVER): An agent workflow. arXiv. https://arxiv.org/pdf/2502.13311

[5] Liu, Q., Zhang, Z., Huang, Z., & Gao, W. (2025). LPITutor: An intelligent tutoring system. PeerJ Computer Science, 11, e2991. https://doi.org/10.7717/peerj-cs.2991

[6] Rozière, B., Gehring, J., Gloeckle, F., et al. (2023). Code Llama: Open foundation models for code. arXiv. https://arxiv.org/abs/2308.12950

[7] Luo, Z., Xu, C., Zhao, P., et al. (2023). WizardCoder: Empowering code large language models. arXiv. https://arxiv.org/abs/2306.08568

[8] D'Andrea, V., Prescod-Weinstein, C., & Mazur, E. (2025). AI tutoring outperforms active learning. Scientific Reports, 15, 17458. https://doi.org/10.1038/s41598-025-97652-6