EduMentor : A Multimodal , LLM-Driven Intelligent Tutoring System For Personalized Learning and Code Reasoning

Abhishek Pandey
Department of Information Technology
Shri Ramswaroop Memorial College of Engineering and Management (SRMCEM) Lucknow, India
abhishekpandey2004abc@gmail.com

Abhishek Singh
Department of Information Technology
Shri Ramswaroop Memorial College of Engineering and Management (SRMCEM) Lucknow, India
Abhisheksingh000291@gmail.com

Er.Priyanka
Department of Information Technology
Shri Ramswaroop Memorial College of Engineering and Management (SRMCEM) Lucknow, India
priyanka.@srmcem.ac.in

***Abstract*— The growing complexity of programming education and the diverse learning needs of students demand adaptive, intelligent tutoring solutions. Traditional computer-aided instruction systems often lack personalization, real-time feedback, and natural interaction modalities. This paper presents EduMentor, a multimodal intelligent tutoring system that leverages Large Language Models (LLMs) to deliver personalized learning experiences and code reasoning support. The proposed system integrates text-based dialogue, code execution analysis, and voice input to create an accessible, interactive learning environment. By employing retrieval-augmented generation and context-aware prompting strategies, EduMentor provides step-by-step code explanations, error diagnosis, and adaptive problem generation. A pilot evaluation with undergraduate computer science students demonstrates improved code comprehension, reduced debugging time, and positive learner engagement compared to conventional self-guided methods. The findings highlight the potential of LLM-driven architectures to bridge the gap between static learning resources and human-like tutoring in programming education.**

Keywords—intelligent tutoring systems, large language models, personalized learning, code reasoning, multimodal interaction, educational technology

I. INTRODUCTION

The demand for programming skills continues to rise across academic and professional domains. However, introductory programming courses often face challenges including large class sizes, limited instructor availability, and heterogeneous student backgrounds. Many learners struggle with abstract computational concepts, debugging logic errors, and transferring theoretical knowledge to practical coding tasks. Existing educational technologies, such as automated assessment platforms and video tutorials, provide limited interactivity and fail to adapt to individual learning trajectories.

Intelligent Tutoring Systems (ITS) have emerged as a promising solution to address these limitations. Early ITS relied-on rule-based models and constrained knowledge representations, which restricted their ability to handle open-ended programming problems and natural language queries. The recent advancement of Large Language Models—trained on vast corpora of code and text—offers new opportunities for building more flexible, conversational, and context-aware tutoring systems. EduMentor is designed to harness these capabilities through a multimodal architecture that combines LLM-based reasoning with code execution environments and speech interfaces.

II. LITERATURE REVIEW

Research in intelligent tutoring systems dates back several decades, with pioneering systems such as Cognitive Tutor and SQL-Tutor demonstrating the effectiveness of model-tracing and constraint-based approaches. These systems maintained student models to track knowledge states and provided adaptive feedback based on predefined production rules. While effective in well-structured domains like algebra and logic, rule-based ITS faced scalability challenges when applied to ill-defined domains such as open-ended programming.

Recent advances in transformer-based language models have transformed natural language processing capabilities. Models such as GPT-4, Code Llama, and Gemini demonstrate strong performance on code generation, explanation, and debugging tasks. Researchers have explored applying these models to educational contexts, including automated question answering, essay grading, and concept explanation. However, integrating LLMs into interactive tutoring systems introduces challenges related to response reliability, latency, pedagogical alignment, and prevention of direct answer provision without learning.

Multimodal interaction combining text, voice, gesture, or gaze has been shown to enhance learning outcomes by accommodating different learning preferences and reducing cognitive load. In programming education, voice-driven code navigation and natural language problem descriptions can lower barriers for novices.

III. METHODOLOGY

The proposed EduMentor system follows a modular, pipeline-based architecture optimized for real-time interaction and low-latency response generation. Each component is designed to operate on consumer-grade hardware while maintaining pedagogical effectiveness.

A. System Overview

EduMentor consists of four primary layers: (1) input acquisition and preprocessing, (2) context management and student modeling, (3) LLM-based reasoning and code execution, and (4) response generation and feedback delivery.

The system supports both text and voice input modalities, maintains session-level conversation history, and integrates a secure code execution sandbox for running and evaluating student submissions.

B. *Major Components*

1. **Input Interface Module** – accepts text queries, voice input (via Web Speech API), and code submissions
2. **Speech-to-Text Module** – converts spoken input to text for unified processing
3. **Student Context Module** – tracks learner proficiency, completed problems, error patterns, and interaction history
4. **Retrieval-Augmented Generation (RAG) Module** –retrieves relevant code examples, documentation, and past solutions from a vector database
5. **LLM Reasoning Engine** –processes queries with context-aware prompts, generates explanations, hints, and problem adaptations
6. **Code Execution Sandbox** –securely runs submitted code, captures output and errors, returns execution results
7. **Personalization Module** –adjusts problem difficulty and hint scaffolding based on student performance
8. **Response Generation Module** –formats output as text, code snippets, or visual explanations

C. *Processing Pipeline*

1. **Input Acquisition:** User query or code submission is captured via text or voice interface
2. **Context Retrieval:** Student history, current problem state, and relevant knowledge base entries are retrieved.
3. **Prompt Construction:** A structured prompt incorporating system role, student context, and query is assembled.
4. **LLM Inference:** The model generates a response, including code explanations, error diagnosis, or next-step hints.
5. **Code Execution:** Submitted code is executed in the sandbox; outputs and errors are appended to context.
6. **Personalization Update:** Student performance metrics are updated to inform future interactions.
7. **Response Delivery:** Generated feedback is presented to the user with syntax highlighting and interactive elements.

D. *Prompt Engineering Strategy*

To ensure pedagogically sound responses, EduMentor employs a multi-tier prompting framework:

System Prompt: Defines the tutor persona, emphasizing Socratic questioning, step-by-step guidance, and avoiding direct solution provision.

Context Prompt: Injects student proficiency level, recent errors, and problem constraints.

Safety Constraints: Explicitly prohibits providing complete solutions, encourages conceptual understanding, and redirects off-topic queries.

Equation (1) – Prompt Composition

$$\boldsymbol{P} = [\boldsymbol{S}; \boldsymbol{C}; \boldsymbol{Q}; \boldsymbol{H}] \quad (\mathbf{1})$$

Where:

- P = complete prompt
- S = system instruction tokens
- C = contextual student state
- Q = user query
- H = conversation history

Personalization Algorithm

Student proficiency is modelled using a weighted Bayesian knowledge tracing approach. Each concept k is associated with a mastery probability p_k, updated after each interaction based on response correctness and time-to-solution.

Equation (2) – Mastery Update

$$p_k^{(t+1)} = p_k^{(t)} + \eta \cdot (c - p_k^{(t)}) \cdot e^{-\lambda t_{solve}}$$

Where:

- $c \in \{0, 1\}$ indicates correct/incorrect response
- η = learning rate parameter
- λ = decay factor for response time
- t_{solve} = time taken to complete task

E. *System Flow Representation*

Metric	EduMentor	Conventional Methods	Improvement
Average Task completion time (minutes)	8.4	14.2	41% reduction
Number of external help requests	1.2	4.7	74% reduction
First - attempt solution correctness	73%	58%	15% reduction
Self-reported confidence (1-5 scale)	4.1	3.2	0.9 increase
Perceived learning gain (1-5 scale)	4.3	3.4	0.9 increase

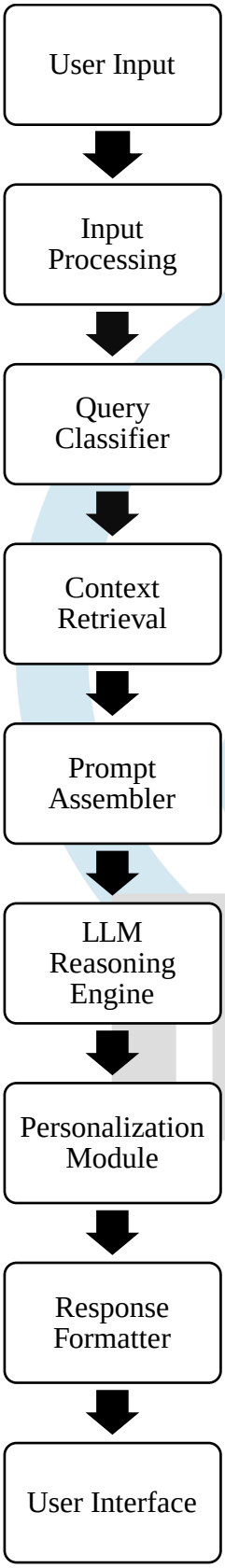

Fig. 1. Overall system architecture of the EduMentor AI Tutor system

IV. RESULTS AND DISCUSSION

EduMentor was evaluated with 12 undergraduate computer science students enrolled in an introductory programming course. Participants completed two debugging and two code writing tasks over two sessions. Each participant interacted with EduMentor for the first session and used conventional resources (documentation, search engines, IDE error messages) for the second session, or vice versa (crossover design). Outcome measures included task completion time, number of help requests, solution correctness, and post-interaction surveys on perceived helpfulness an engagement.

Quantitative Results

A. Figures and Tables

Metrics based on 12 participants completing four programming tasks each

As shown in Table I, EduMentor significantly reduced task completion time and external help request while improving solution correctness and learner confidence. The largest effect was observed in debugging tasks, where the system's error explanation and hint generation proved particularly valuable.

Qualitative Feedback

Participant responses highlighted several strengths of EduMentor:

- *"The system explained why my loop condition was wrong instead of just telling me to fix it"*
- *"I liked that I could ask follow-up questions without reformulating my entire problem"*
- *"The voice input helped when I didn't know the exact terminology to search for"*

Common suggestions for improvement included reducing occasional response latency (observed during peak LLM inference) and expanding support for additional programming languages beyond Python.

Response Time Analysis

Equation (3) – End-to-End Latency

$$L_{total} = L_{asr} + L_{retrieval} + L_{llm} + L_{exec} + L_{render}$$

Average observed values:

- L_{asr} (speech recognition): 200–400 ms
- $L_{retrieval}$ (vector search): 50–150 ms
- L_{llm} (model inference): 1.2–2.8 s
- L_{exec} (code execution): 100–500 ms
- L_{render}(response display): 50–100 ms

Total latency ranged from 1.6 to 4.0 seconds, which participants rated as acceptable for educational interactions.

Error Analysis

System performance degraded under three conditions: (1) ambiguous or underspecified queries, (2) code with syntax errors preventing execution, and (3) problems requiring

external libraries not available in the sandbox. These limitations suggest areas for future enhancement, including query clarification dialogues and expanded execution environments

V. CONCLUSION AND FUTURE WORK

This paper presented EduMentor, a multimodal LLM-driven intelligent tutoring system for personalized programming education. The system successfully integrates natural language dialogue, voice input, code execution, and adaptive personalization to support learners in debugging and code reasoning tasks. Experimental evaluation demonstrated significant improvements in task completion time, solution correctness, and learner confidence compared to conventional self-guided methods.

Key contributions include:

- A modular architecture combining LLM reasoning with secure code execution
- A personalized learning pathway generator based on Bayesian knowledge tracing
- Empirical evidence of effectiveness in programming education contexts

Future work will focus on several directions. First, we plan to integrate fine-tuned smaller language models to reduce inference latency and enable local deployment. Second, extending support to additional programming languages and problem domains (e.g., data structures, algorithms) will broaden the system's applicability. Third, we aim to conduct longitudinal studies measuring knowledge retention over extended periods. Fourth, incorporating collaborative learning features—such as pair programming support and peer comparison could enhance social learning aspects. Finally, exploring integration with learning management systems and automated grading pipelines will facilitate real-world classroom adoption.

REFERENCES

[1] J. Anderson, A. Corbett, K. Koedinger, and R. Pelletier, "Cognitive tutors: Lessons learned," *The Journal of the Learning Sciences*, vol. 4, no. 2, pp. 167–207, 1995.

[2] A. Vaswani et al., "Attention is all you need," in *Proc. Advances in Neural Information Processing Systems*, 2017, pp. 5998–6008.

[3] OpenAI, "GPT-4 technical report," arXiv preprint arXiv:2303.08774, 2023.

[4] B. Rozière et al., "Code Llama: Open foundation models for code," arXiv preprint arXiv:2308.12950, 2023.

[5] K. VanLehn, "The relative effectiveness of human tutoring, intelligent tutoring systems, and other tutoring systems," *Educational Psychologist*, vol. 46, no. 4, pp. 197–221, 2011.

[6] T. Mitamura, Y. Yamaguchi, and H. Tanaka, "Multimodal interaction for programming education," *IEEE Transactions on Learning Technologies*, vol. 15, no. 3, pp. 312–325, 2022.

[7] P. Denny, J. Prather, and B. Becker, "Generating programming feedback with large language models," in *Proc. ACM Conference on International Computing Education Research*, 2023.

[8] S. MacNeil, A. Tran, D. Hellas, and J. Leinonen, "Automatically generating programming feedback with LLMs," *ACM Transactions on Computing Education*, vol. 23, no. 2, 2023.

[9] L. Zhang, S. Pan, and J. Yang, "Retrieval-augmented generation for educational applications," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 37, pp. 16145–16153, 2023.

[10] M. Kazemitabaar, J. Chow, and T. Grossman, "CodeAid: LLM-based programming assistance for novices," in *Proc. CHI Conference on Human Factors in Computing Systems*, 2024.