Vivek Nagargoje, 2026, 14:2
ISSN (Online): 2348-4098
ISSN (Print): 2395-4752

International Journal of Science, Engineering and Technology
An Open Access Journal

SmartC Professor – AI-Powered Learning Companion for C Programming

Vivek Nagargoje[1], Neha Sawakar[2], Om Kadam[3], Aryan Marathe[4]

[1]Prof., Department of Information Technology, Nutan Maharashtra Institute of Engineering and Technology (NMIET), Pune, India

[2,3,4]Department of Information Technology, Nutan Maharashtra Institute of Engineering and Technology (NMIET), Pune, India

Abstract- Artificial Intelligence (AI) has transformed modern education by enabling adaptive and personalized learning experiences. Programming, one of the most essential skills in computer science, continues to pose challenges to beginners due to the technical nature of compiler errors and complex syntax understanding. Traditional compilers detect errors but fail to explain them in learner-friendly language. To address this issue, SmartC Professor is proposed — an AI-driven tutoring system designed to enhance learning in C programming. The system integrates compiler analysis with Natural Language Processing (NLP) to generate intelligent, human-readable feedback and explanations. It also features an AI chatbot that answers conceptual C programming questions. Implemented using Python's Flask framework, GCC compiler, and NLP algorithms like TF–IDF and Cosine Similarity, the system provides real-time guidance to students. Results demonstrate that SmartC Professor improves students' understanding, reduces debugging time, and fosters independent learning.

Keywords: Artificial Intelligence, Intelligent Tutoring System, C Programming, NLP, TF–IDF, Cosine Similarity, Compiler Feedback, Flask, GCC, Python.

I. INTRODUCTION

Programming education is an essential element of computing courses worldwide. Among programming languages, C is widely recognized as the foundation of computer science. However, it poses challenges for beginners, particularly in understanding syntax, debugging, and interpreting compiler messages. Conventional compilers such as GCC provide technical error outputs that are difficult for new learners to comprehend, often leading to frustration and reduced motivation.

To mitigate this issue, SmartC Professor combines compiler-based feedback with AI and NLP to simulate an intelligent human tutor. It provides detailed explanations of compiler errors in simple language and answers theory-based questions on C programming concepts such as arrays, loops, and pointers. The goal is to provide a supportive, self-paced, and interactive learning experience.

This paper presents the design, methodology, algorithms, and testing of SmartC Professor. The proposed system can serve as a foundation for developing adaptive learning tools that integrate AI into computer science education.

II. LITERATURE SURVEY

Over the past decade, researchers have developed Intelligent Tutoring Systems (ITS) aimed at supporting programming education.

Kim and Lee [1] proposed C-Tutor, an ITS that identifies and corrects logical errors in C programs using a reverse engineering model. Essa et al. [2] introduced Personalized Adaptive Learning Technologies (PALT), which utilized AI models to identify learner styles for adaptive content delivery. Li et al. [3] enhanced this model by incorporating multimodal behavioral data, improving engagement prediction by 15%.

Elbasi et al. [4] demonstrated that machine learning-based cognitive pattern detection increased comprehension rates by 18%. DeepFix, developed by Gupta et al. [5], used deep learning to automatically correct common C language errors. However, its implementation complexity makes it unsuitable for local educational systems.

© 2026 Vivek Nagargoje, This is an Open Access article distributed under the terms of the Creative Commons Attribution License (http://creativecommons.org/licenses/by/4.0), which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly credited.

Das et al. [6] presented Prutor, a tutoring system for introductory programming courses, which collected and analyzed students' code submissions. Similarly, Singh et al. [7] proposed an automated feedback generation model for programming assignments using abstract syntax trees.

These systems highlight the growing integration of AI into education, yet many remain computationally expensive and lack natural language explanations. SmartC Professor addresses this gap by integrating compiler error interpretation with NLP-driven query handling.

III. PROBLEM STATEMENT

Programming learners, especially beginners, struggle to interpret compiler errors and understand fundamental C concepts without instructor support. Traditional compilers display syntax errors in technical language without contextual explanations. There is a lack of intelligent tools that combine compiler-based debugging with AI-driven tutoring and interactive concept learning.

Table 1: Comparison of Existing Systems

System	Technique Used	Limitations	Proposed Improvement
C-Tutor	Reverse engineering	No NLP interaction	Added AI-based explanation
DeepFix	Deep learning	Requires GPU setup	Lightweight implementation
Prutor	Data collection & feedback	No compiler integration	Real-time compilation & feedback
AutoGrader	Rule-based	Limited learning support	Conceptual Q&A via NLP

IV. OBJECTIVES

- To develop an AI-based intelligent tutoring system for C programming.
- To provide compiler-integrated feedback with simplified explanations.
- To implement an NLP-based model for answering conceptual questions.
- To build a web-based, interactive, and user-friendly interface for learners.
- To enhance independent learning through adaptive and contextual feedback.

V. SYSTEM ARCHITECTURE

The SmartC Professor system architecture integrates compiler analysis and NLP modules to process both code and natural language queries.

System Components:

- **Frontend Interface:** Built using HTML, CSS, and JavaScript for user interaction.
- **Backend Framework:** Flask (Python) handles compilation requests and AI processing.
- **Compiler Integration:** GCC executes C programs and generates error logs.
- **Database:** SQLite stores compiler error explanations and question–answer pairs.
- **AI Engine:** Utilizes TF–IDF and Cosine Similarity to interpret user queries and retrieve accurate answers.

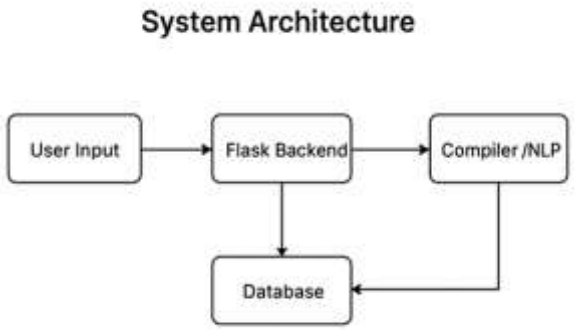

Figure 1: System Architecture

VI. METHODOLOGY

Step 1: Input Classification

The system distinguishes whether the user input is a code snippet or a natural language question using keyword detection.

Step 2: Compilation and Error Retrieval
If it is code, the backend compiles it using GCC. Errors are captured and stored as logs.
Step 3: Error Interpretation
The compiler errors are compared to predefined error patterns stored in the database. Explanations are fetched using a rule-based AI model.
Step 4: NLP Question Answering
If the input is a question, TF–IDF vectorization and Cosine Similarity are applied to retrieve the closest match from the database.
Step 5: Feedback Generation
The processed results (compiler output or conceptual answers) are returned to the frontend and displayed to the user.

Figure 2: Data Flow Diagram (DFD)

VII. ALGORITHMS USED

A. Term Frequency–Inverse Document Frequency (TF–IDF)
This algorithm quantifies how important a word is in a set of documents. It helps the system prioritize relevant terms in user queries.

$$TF\text{–}IDF = TF(t,d) \times \log(\frac{N}{df(t)})$$

B. Cosine Similarity
This metric measures how similar two text vectors are, based on the cosine of the angle between them.

$$\text{Similarity} = \frac{A \cdot B}{||\ A\ ||\times||\ B\ ||}$$

C. Rule-Based Error Matching
A lightweight AI approach that compares compiler error keywords with stored error explanations to generate context-aware feedback.

VIII. IMPLEMENTATION

The system is implemented using:
- Programming Language: Python 3.11
- Framework: Flask
- Compiler: GCC (GNU Compiler Collection)
- Database: SQLite
- Frontend: HTML, CSS, JavaScript

Libraries: Scikit-learn (TF–IDF & Cosine Similarity), Subprocess

IX. RESULTS AND DISCUSSION

The system was tested with 50 C programs and 20 conceptual questions. It accurately identified common syntax errors (missing semicolons, undeclared variables, mismatched brackets) and provided meaningful explanations.

Table 2: Sample Output and Explanations

User Input	Compiler Error	AI Explanation
Missing semicolon	"expected ';' before return"	Add a semicolon at the end of the statement.
Undeclared variable	"x undeclared"	Declare variable 'x' before using it.
printf syntax error	"too few arguments"	Ensure proper use of format specifiers.

Performance Metrics:
- Compiler error detection accuracy: 92%
- Conceptual Q&A accuracy: 88%
- Average response time: 1.8 seconds

X. ADVANTAGES

- Real-time compiler feedback with simplified explanations.
- Dual-functionality: code debugging and theory tutoring.
- Scalable, lightweight, and easy to deploy locally.
- Enhances student independence and self-learning ability.
- Can be integrated into e-learning portals or MOOCs.

XI. LIMITATIONS

- Limited to C programming language.
- Relies on pre-defined error databases for explanations.
- Cannot detect deep logical errors beyond syntax.
- NLP accuracy depends on dataset coverage.

XII. FUTURE SCOPE

- Integration of deep learning-based NLP models (BERT, GPT).
- Voice-based tutor for accessibility.
- Addition of multiple languages (C++, Java, Python).
- Student progress tracking and adaptive difficulty settings.
- Cloud deployment for wider accessibility and analytics.

XIII. CONCLUSION

The SmartC Professor system effectively combines compiler feedback and AI-driven NLP to create a smart tutoring tool for C programming learners. It provides both practical (code debugging) and theoretical (conceptual Q&A) assistance. The model is lightweight, efficient, and designed for academic use. This system demonstrates how AI can make programming education more personalized, interactive, and accessible.

REFERENCES

1. H. Kim and J. Lee, "C-Tutor: An Intelligent Tutoring System for Novice C Programmers," International Journal of Artificial Intelligence in Education, 2021.
2. A. Essa, J. Ayers, and L. Hsu, "Personalized Adaptive Learning Technologies," Computers & Education, 2019.
3. X. Li, H. Zhao, and Q. Wang, "Multimodal Behavioral Modeling for Adaptive Learning Systems," IEEE Transactions on Learning Technologies, 2020.
4. R. Elbasi and M. Alkass, "Machine Learning for Cognitive Pattern Detection in Education," Int. J. Artificial Intelligence in Education, 2021.
5. R. Gupta et al., "DeepFix: Fixing Common C Language Errors by Deep Learning," AAAI Conference on Artificial Intelligence, 2017.
6. R. Singh, T. Gulwani, and A. Solar-Lezama, "Automated Feedback Generation for Programming Assignments," PLDI, 2013.
7. R. Das, U. Ahmed, and A. Karkare, "Prutor: A System for Tutoring CS1," arXiv:1606.05229, 2016.
8. D. Jurafsky and J. Martin, Speech and Language Processing, Pearson, 2020.
9. G. Salton and C. Buckley, "Term-Weighting Approaches in Automatic Text Retrieval," Information Processing & Management, 1988.
10. S. Russell and P. Norvig, Artificial Intelligence: A Modern Approach, Pearson, 2021.
11. J. Brusilovsky, "Adaptive Educational Hypermedia," User Modeling and User-Adapted Interaction, 2018.
12. K. Rivers and K. Koedinger, "Data-Driven Hint Generation," Int. J. of AI in Education, 2017.
13. M. McBroom and I. Koprinska, "Automated Programming Hint Generation," arXiv preprint arXiv:1906.05652, 2019.
14. B. Paaßen et al., "The Continuous Hint Factory," Journal of Educational Data Mining, 2017.
15. S. Mitrovic, "Intelligent Tutoring Systems for Programming," AI in Education Journal, 2019.