PyPAF:Python Program,Algorithm and Flowchart Generator

Fathima Nourin S U,Haritha Krishna R,Hiba Hussain,Surya Ann Joshy

Guide: Mrs.Fathima N

Department of Computer Engineering,

College of Engineering, Cherthala, Kerala, India

Email: *fathimanourin369@gmail.com,harithapisharody@gmail.com,mail.hibahussain@gmail.com,joshypk@gmail.com*

***Abstract*—In the evolving landscape of programming education, automation tools play a crucial role in enhancing the teaching and learning experience. This paper presents an intelligent code generation and evaluation system that transforms user input into structured pseudocode, serving as the foundation for generating Python code, algorithms, and flowcharts. The system automates code creation while ensuring adherence to coding standards and algorithmic correctness. Additionally, a code evaluation feature is integrated to assist educators in assessing student-submitted code by comparing it with the system-generated code. This comparison provides valuable insights into code accuracy and efficiency, facilitating a structured evaluation of students programming skills. By combining automation with structured evaluation methodologies, this system serves as a powerful tool for improving programming education and assessment.**

***Index Terms*—Pseudocode generation, Code generation, Flowchart generation, Algorithm generation, Code evaluation, Black-box AI, Hugging Face model.**

I. INTRODUCTION

In the evolving landscape of digital education, the ability to transform ideas into functional code, structured algorithms, and visual representations plays a crucial role in programming and computational thinking. However, individuals with limited programming expertise often struggle to convert their logical concepts into executable solutions. This challenge affects both students, who require guided learning tools, and educators, who face difficulties in efficiently assessing programming assignments.

To bridge this gap, this project introduces an automation-driven web application that facilitates programming education by generating pseudocode, structured algorithms, flowcharts, and Python code from user-provided queries. The system enables students to input logic in simple language, which is then parsed and converted into meaningful outputs, aiding in their understanding of programming concepts. Additionally, an automated grading system is integrated to evaluate code correctness and similarity, allowing educators to assess student submissions with greater accuracy and efficiency.

The application enhances interactive learning by automating pseudocode-to-algorithm conversion, flowchart generation, and code generation, reducing the manual effort required in programming education. Furthermore, by simplifying complex computational processes, the system ensures that both beginners and experienced users can engage with programming concepts more effectively. By leveraging automation and AI-driven techniques, the project aims to improve accessibility, streamline programming assessments, and foster a more inclusive and engaging approach to programming education.

II. PROBLEM STATEMENT

Programming education presents significant challenges for both students and educators, particularly in translating problem statements into structured algorithms, flowcharts, and executable code. Many students struggle with developing algorithmic thinking, which is essential for problem-solving in programming. Educators, on the other hand, face difficulties in effectively demonstrating programming concepts in a structured and interactive manner.

Additionally, the evaluation of student-submitted code for correctness, efficiency, and similarity to expected solutions is a labor-intensive and time-consuming process. Manual assessment requires significant effort, making it challenging for educators to provide timely and consistent feedback to students.

To address these challenges, this project proposes the development of a user-friendly automation tool designed to convert user-provided prompts into pseudocode, structured algorithms, flowcharts, and Python code. The tool functions as an educational aid, facilitating the teaching of programming concepts and algorithm design. Furthermore, the system includes an automated grading mechanism that evaluates student submissions by comparing them against system-generated solutions, ensuring efficient and accurate assessment.

By automating key aspects of programming education, the proposed system aims to enhance learning outcomes, improve instructional efficiency, and reduce the manual workload associated with code evaluation.

III. LITERATURE SURVEY

Several existing research studies and technologies have laid the foundation for the development of automated code generation and flowchart-based programming tools. These works explore key components such as source code parsing, model-driven development, visual programming, and AI-assisted code generation, all of which contribute to the design and functionality of this project.

- **An Extensible Approach to Generate Flowcharts from Source Code** [1]: This study by Damitha D. Karunarathna and Nasik Shafeek (2018) presents a method for automatically generating flowcharts from source code using a two-component architecture: a Frontend Translator that parses the code into an Abstract Syntax Tree (AST) and a Backend Component that converts the AST into a visual representation using Graphviz. This extensible approach supports multiple languages through modular front-end translators, highlighting the feasibility of automated flowchart generation.
- **Code Similarity Detection using AST and Textual Information** [2]: This study by Wu Wena, Xiaobo Xueb, Ya Lia, Peng Gua, and Jianfeng Xub o (2019) presents a hybrid method for plagiarism detection by integrating text similarity using Simhash and structural similarity using AST edit distance with the Zhang-Shasha algorithm. This approach effectively detects plagiarism techniques such as variable renaming, code restructuring, and order modification. The method provides a comprehensive similarity evaluation that improves over purely text-based or purely structural approaches, making it a tool for plagiarism detection in programming courses.
- **Automatic Code Generation for C and C++ Programming** [3]: A tool developed by Sanika Patade, Pratiksha Patil, Ashwini Kamble, and Prof. Madhuri Patil (2021) allows users to create C and C++ code by designing visual flowcharts, which are then converted into executable programs. This research highlights the advantages of reducing syntax complexity and providing an intuitive interface for programming, influencing our approach to integrating visual programming techniques for code generation.
- **Leveraging Pre-Trained Language Models for Code-Generation** [4]: Ahmed Soliman, Samir Shaheen, and Mayada Hadhoud (2024) investigate the use of pre-trained transformers such as BERT, RoBERTa, and ELECTRA for code generation. Their model fine-tunes these architectures on CoNaLa and DJANGO datasets using Marian as a decoder, achieving improved BLEU scores and code quality. The findings underscore the potential of transformer-based AI models in generating structured code, which aligns with the integration of NLP models in modern development tools.
- **Intelligent Code Generation with Natural Language Processing and OpenAI's GPT-4 API** [5]: Brain H.Hough (2024) introduce Evolve, an AI-based software development system powered by GPT-4. By incorporating an iterative feedback loop for testing, compilation, and repository management, Evolve demonstrates how AI can continuously refine and optimize software development. The approach aligns with ongoing advancements in AI-driven code assistance for software engineering.
- **Code Generation from Flowchart using Optical Character Recognition and Large Language Model** [6]: Aryaman Darda and Reetu Jain (2024) explores Optical Character Recognition (OCR) and Llama-2-Chat for extracting text from flowcharts and converting it into Python code. The system improves accessibility and reduces manual coding effort, emphasizing the growing role of AI and OCR in automating programming tasks.

These studies collectively validate the feasibility of developing automated, AI-assisted, and visual programming-based code generation systems. Their methodologies and findings significantly influenced the architectural and functional decisions made in this project.

IV. Proposed System

The proposed system automates the transition from natural language problem descriptions to structured programming artifacts, including pseudocode, algorithms, flowcharts, and Python code. It is designed for both students and teachers, streamlining code generation, evaluation, and feedback.

- Student Module: Converts problem descriptions into pseudocode, algorithm, flowchart, and Python code, allowing students to execute and review outputs.
- Teacher Module: Generates reference solutions and evaluates student submissions by comparing them against the reference code using AST-based similarity metrics

The system architecture includes the following key components:

- **User Interface Layer**: The User Interface (UI) Layer is a web-based platform where students and teachers interact with the system. Students can input natural language descriptions, and the system generates pseudocode, algorithms, flowcharts, and executable Python code. Teachers can provide problem statements, generate reference solutions, and review student submissions. The UI ensures real-time responses, supports code preview, flowchart display, and report generation, making it easy for users to navigate and use the system effectively.
- **Processing Layer**:The Processing Layer converts natural language input into structured programming formats. It uses Natural Language Processing (NLP) models to analyze the input, extract key programming concepts, and generate pseudocode. Machine learning models, including pre-trained NLP transformers, help interpret the input accurately. This structured output forms the basis for generating flowcharts and executable code. The accuracy of this layer is essential for producing reliable outputs.
- **Flowchart Visualization Layer**: The Flowchart Visualization Layer converts the algorithm representation into a graphical flowchart. It helps students visualize the logic before coding. The system automatically maps decision points, loops, and operations into flowchart symbols. The final flowchart is stored as an image, allowing users to download or view it. This feature makes understanding algorithms easier and improves learning.
- **Computation and Evaluation Layer**: The Computation and Evaluation Layer is responsible for code generation and automated assessment. It also evaluates student

submissions by comparing their code with the teacher's reference solution using Abstract Syntax Tree (AST) analysis. This ensures objective and consistent grading, identifies errors and deviations, and reduces the teacher's workload.
- **Feedback System Layer**: The Feedback System Layer helps evaluate student submissions by generating detailed reports. It highlights errors such as syntax mistakes , making it easier for students to improve their code. It also checks algorithm similarity, comparing the student's approach with the reference solution to find differences. The system verifies how many test cases are passed, ensuring the code works correctly. Based on these factors, it assigns grades, making the evaluation process faster and more accurate for teachers while helping students learn from their mistakes.

V. SYSTEM DESIGN

A. Input Processing Module

The User Selection Module determines whether the system is being accessed by a student or a teacher. If a student accesses the system, they are guided through the pseudocode generation, algorithm conversion, and flowchart creation processes. Teachers, on the other hand, gain access to evaluation tools that allow them to review and assess student submissions. This module ensures that each user type experiences a tailored interface optimized for their specific needs.

B. Pseudocode Generation Module

The Pseudocode Generation Module is responsible for converting natural language descriptions into pseudocode using the BlackBox API. The system sends the user input to the API, retrieves the generated pseudocode, and stores it for further processing. This module ensures that user inputs are transformed into structured programming logic, serving as the foundation for algorithm and code generation. The implementation is handled using Python and the Requests library for seamless API communication.

C. Algorithm Generation Module

The Algorithm Generation Module takes the generated pseudocode and converts it into a structured step-by-step algorithm. This ensures that users can easily understand the logical flow of the solution before proceeding to code implementation. The module employs custom transformation logic in Python to extract well-structured algorithmic steps from the pseudocode, maintaining clarity and consistency.

D. Flowchart Visualization Module

The Flowchart Visualization Module extends the capabilities of the system by converting pseudocode into a graphical representation of logic flow. After parsing the pseudocode, it classifies logical statements into nodes such as decision points, input/output steps, and processing steps. A flowchart construction layer arranges these classified nodes based on control flow logic, creating a structured visual representation. This module enhances the learning experience by enabling students to comprehend the flow of execution in an intuitive manner.

E. Code Generation Module

The Code Generation Module is responsible for converting pseudocode into executable Python code. The generated pseudocode is passed to the Hugging Face model, which translates it into syntactically correct and functional Python code while ensuring adherence to programming standards. The generated code is then presented to the user for execution and verification. To enhance usability, the interface includes a Copy button, allowing users to quickly copy the generated code for external use, and a Run button, enabling them to execute the code directly within the platform. This functionality provides immediate feedback on the correctness and functionality of the generated code, ensuring a seamless and efficient coding experience.

F. Code Evaluation Module

The Code Evaluation Module is designed to assess student-submitted Python programs by comparing them with a reference solution. A reference code is first generated using a Hugging Face model based on a given problem statement. The submitted code is then evaluated for similarity using Levenshtein Distance and SequenceMatcher, ensuring structural and functional accuracy. The module also utilizes Abstract Syntax Tree (AST) parsing to analyze the code's structure and includes a timeout mechanism to prevent infinite loops or excessive execution time. The final evaluation considers functional correctness, structural similarity, and efficiency, providing an automated and objective assessment.

VI. SYSTEM ARCHITECTURE

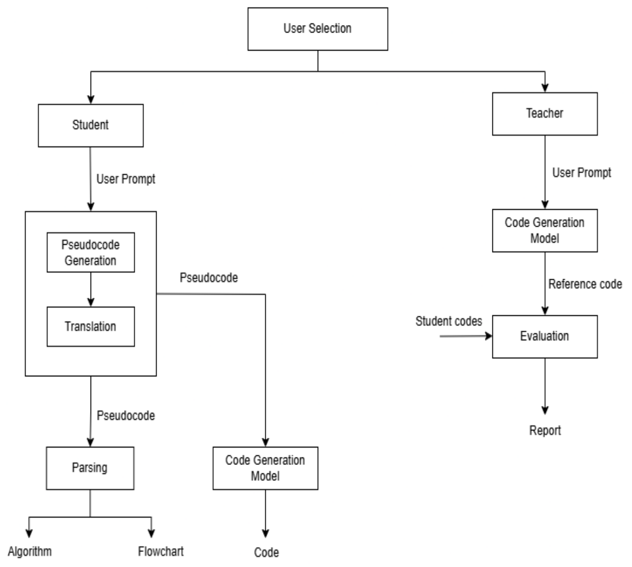


Fig. 1. Architecture

VII. Implementation Results

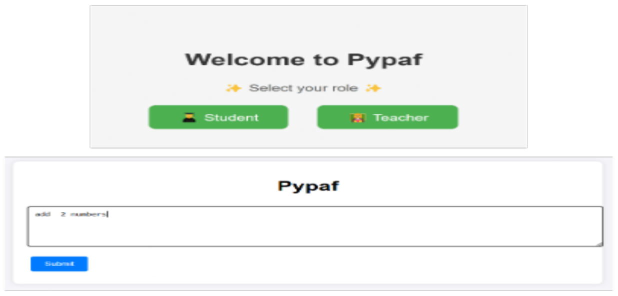
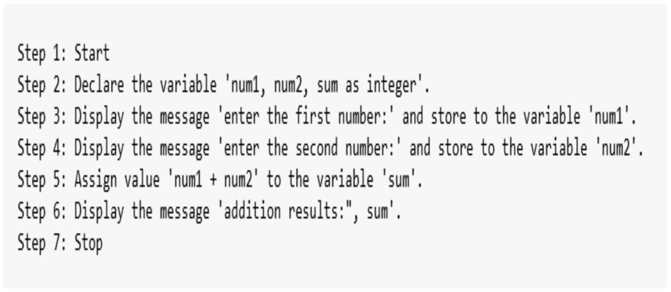


Fig. 2. User Interface - Student Input



```
Step 1: Start
Step 2: Declare the variable 'num1, num2, sum as integer'.
Step 3: Display the message 'enter the first number:' and store to the variable 'num1'.
Step 4: Display the message 'enter the second number:' and store to the variable 'num2'.
Step 5: Assign value 'num1 + num2' to the variable 'sum'.
Step 6: Display the message 'addition results:", sum'.
Step 7: Stop
```

Fig. 3. Generated Algorithm

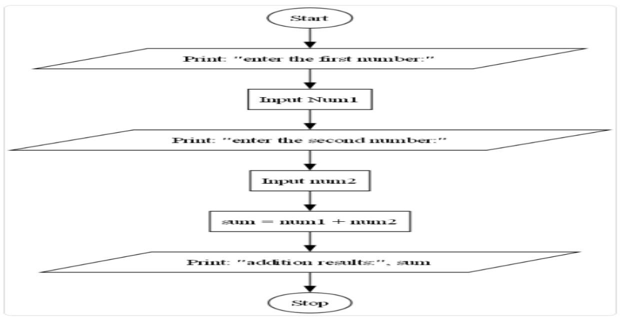
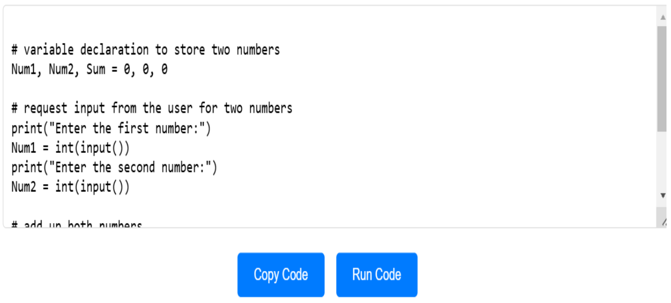

Start
Print: "enter the first number:"
Input Num1
Print: "enter the second number:"
Input num2
sum = num1 + num2
Print: "addition results:", sum
Stop

Fig. 4. Generated Flowchart

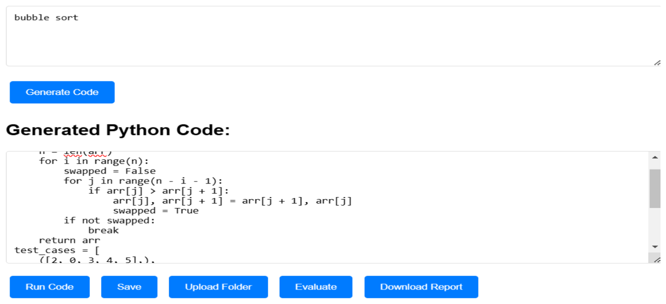


```
# variable declaration to store two numbers
Num1, Num2, Sum = 0, 0, 0

# request input from the user for two numbers
print("Enter the first number:")
Num1 = int(input())
print("Enter the second number:")
Num2 = int(input())

# add up both numbers
```

Fig. 5. Generated Python Code

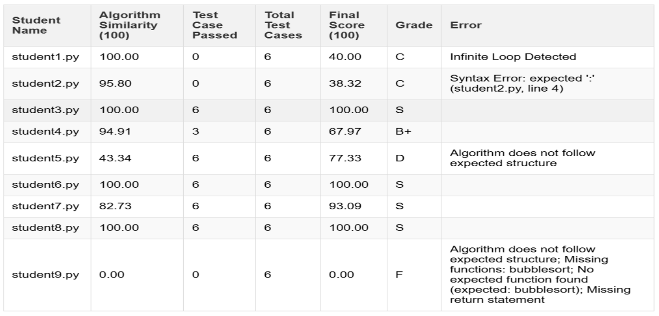


```
bubble sort
```

Generated Python Code:

```
    n = len(arr)
    for i in range(n):
        swapped = False
        for j in range(n - i - 1):
            if arr[j] > arr[j + 1]:
                arr[j], arr[j + 1] = arr[j + 1], arr[j]
                swapped = True
        if not swapped:
            break
    return arr
test_cases = [
    ([2, 0, 3, 4, 5],),
```

Fig. 6. Teacher Input



Student Name	Algorithm Similarity (100)	Test Case Passed	Total Test Cases	Final Score (100)	Grade	Error
student1.py	100.00	0	6	40.00	C	Infinite Loop Detected
student2.py	95.80	0	6	38.32	C	Syntax Error: expected ':' (student2.py, line 4)
student3.py	100.00	6	6	100.00	S	
student4.py	94.91	3	6	67.97	B+	
student5.py	43.34	6	6	77.33	D	Algorithm does not follow expected structure
student6.py	100.00	6	6	100.00	S	
student7.py	82.73	6	6	93.09	S	
student8.py	100.00	6	6	100.00	S	
student9.py	0.00	0	6	0.00	F	Algorithm does not follow expected structure; Missing functions: bubblesort; No expected function found (expected: bubblesort); Missing return statement

Fig. 7. Evaluation Report

VIII. Conclusion and Future Scope

The proposed system presents an efficient and intelligent approach to automatic code generation, algorithm creation, and flowchart visualization, simplifying the programming process for students and educators. By automating the conversion of natural language input into structured pseudocode, algorithms, flowcharts, and executable code, the system enhances learning and productivity. The automated evaluation mechanism ensures accurate and unbiased grading, reducing the manual workload for educators while providing students with structured feedback. This tool serves as a valuable resource for learning, assessment, and improving coding skills, offering a seamless and objective evaluation process.

Looking ahead, several advancements can enhance the system's functionality:

- **Multi-Language Support**: Expanding beyond Python to support multiple programming languages, enabling a broader user base.
- **Function-Based Flowcharts**: Enhancing flowchart generation to accommodate modular and function-based structures, improving representation of complex programs.
- **AI-Driven Debugging and Optimization**: Integrating intelligent debugging tools for error detection, real-time recommendations, and code optimization, refining accuracy and efficiency.
- **Support for Advanced Algorithms**: Strengthening capabilities to handle complex data structures and algorithms,

making the system suitable for advanced programming challenges.

- **Cloud-Based Collaboration**:Enabling real-time collaboration and online platform integration, allowing users to share and work on code remotely.
- **Improved Code Evaluation**: Enhancing accuracy in code assessment by refining syntax and logical error detection, ensuring more fair and liberal grading by distinguishing between minor syntax errors and fundamental logic flaws.

With these enhancements, the system has the potential to become a comprehensive tool for automated code generation and algorithmic learning. By continuously evolving to incorporate new technologies and features, it will empower students and educators alike, streamlining the coding process and fostering a more effective learning experience.

REFERENCES

[1] D. D. Karunarathna and N. Shafeek, ”An Extensible Approach to Generate Flowcharts from Source Code,” *International Journal of Research - Granthaalayah*, vol. 6, no. 9, pp. 505-519, 2018. https://doi.org/10.5281/zenodo.1465019.

[2] W. Wu, X. Xue, Y. Li, P. Gu, and J. Xu, ”Code Similarity Detection using AST and Textual Information,” *International Journal of Performability Engineering*, vol. 15, no. 10, p. 2683, 2019. https://doi.org/10.23940/ijpe.19.10.p14.26832691.

[3] S. Patade, P. Patil, A. Kamble, and M. Patil, ”Automatic Code Generation for C and C++ Programming,” *International Research Journal of Engineering and Technology (IRJET)*, vol. 8, no. 5, May 2021.

[4] A. Mastropaolo, L. Pascarella, E. Guglielmi, M. Ciniselli, S. Scalabrino, R. Oliveto, and G. Bavota, ”On the Robustness of Code Generation Techniques: An Empirical Study on GitHub Copilot,” arXiv, Feb. 2023. 10.48550/arXiv.2302.00438.

[5] A. Soliman, S. Shaheen, and M. Hadhoud, ”Leveraging Pre-trained Language Models for Code Generation,” *Complex Intelligent Systems*, vol. 10, pp. 3955–3980, 2024. https://doi.org/10.1007/s40747-024-01373-8.

[6] A. Darda and R. Jain, ”Code Generation from Flowchart Using Optical Character Recognition and Large Language Model,” 2024.