PyPAF: Python Program, Algorithm, and Flowchart Generator

GROUP - 8
Fathima Nourin S.U (CEC21CS045)
Haritha Krishna R (CEC21CS050)
Hiba Hussain (CEC21CS051)
Surya Ann Joshy (CEC21CS095)

GUIDED BY:
Mrs.Fathima N
DEPARTMENT OF COMPUTER ENGINEERING
COLLEGE OF ENGINEERING CHERTHALA

April 10, 2025

INDEX

1. INTRODUCTION
2. PROBLEM STATEMENT
3. OBJECTIVE
4. LITERATURE SURVEY
5. PRODUCT FUNCTIONS
6. SOFTWARE REQUIREMENTS
7. HARDWARE REQUIREMENTS
8. SYSTEM DESIGN
9. DETAILED DESIGN
10. IMPLEMENTATION & RESULT
11. CONCLUSION
12. REFERENCES

INTRODUCTION

- Coding is now a key skill in schools and colleges, but many students find it difficult to learn and apply programming concepts effectively.
- Teachers often spend hours manually reviewing code during lab exams, which can be time-consuming and challenging to grade fairly.
- Our project aims to simplify the learning process for students by offering guided support and structured assistance in writing and improving their coding skills.
- At the same time, it helps teachers evaluate student work more efficiently, ensuring accurate assessments and saving valuable time during practical evaluations.

PROBLEM STATEMENT

- Students struggle to convert their ideas into working code due to lack of step-by-step guidance.
- Existing tools mainly convert finished code into flowcharts, offering little help during the initial coding process.
- Teachers face challenges in quickly and fairly evaluating multiple student submissions during lab exams.
- There is no tool that connects ideas, flowcharts, and code both ways, making learning and assessment less effective.

OBJECTIVE

- To support learning by enabling students to convert their natural language descriptions into algorithms, flowcharts, and executable code.
- Unlike existing systems that rely on reverse strategies, this tool guides students from idea to implementation step-by-step.
- To assist teachers by automatically generating reference code from prompts and evaluating student submissions using test cases for fair and efficient grading.

LITERATURE SURVEY

LITERATURE SURVEY 1

AN EXTENSIBLE APPROACH TO GENERATE FLOWCHARTS FROM SOURCE CODE (September 2018)[1]

Author : Damitha D Karunarathna, Nasik Shafeek

- Flowcharts make code easier with clear pictures.
- The method uses a tree in three steps, tested on PHP.
- Front-end makes Abstract Syntax Tree(AST), middle writes Dot language, back-end draws flowcharts from PHP.

Advantages:

- Separate components (front-end, middle, back-end) allow independent updates or replacements.
- Uses existing tools like Graphviz, reducing development effort.

Disadvantages:

- Requires understanding of compiler design and AST for extension.
- Generated Dot Language may create cluttered flowcharts.
- Lacks UI.

LITERATURE SURVEY 2

Code Similarity Detection Using AST and Textual Information (2019))[2]
Author: Wu Wen, Xiaobo Xue, Ya Li, Peng Gu, and Jianfeng Xu

- Analyzes code similarity using both text-based and AST-based features.
- Applies SimHash after code normalization to generate fingerprints.
- Extracts AST and computes Zhang-Shasha edit distance, then combines both scores using a weighted formula.

Advantages:

- Effectively detects various forms of code plagiarism.
- Considers both text similarity and code structure similarity.

Disadvantages:

- Lacks a ready-to-use implementation,its more of an idea than a tool.
- Does not evaluate code functionality or runtime correctness.

LITERATURE SURVEY 3

Automatic Code Generation for C and C++ Programming (May 2021)[3]
Author : Sanika Patade, Pratiksha Patil, Ashwini Kamble, Prof. Madhuri Patil

* Automates C/C++ code generation from flowcharts to simplify programming for beginners and visual learners.
* Reduces manual coding complexity by converting user-drawn flowcharts into algorithms and executable programs.

Advantages:

* Simplifies coding, no syntax memorization.
* Supports beginners with integrated save/compile/run features.

Disadvantages:

* Requires user interaction for every step.

LITERATURE SURVEY 4

On the Robustness of Code Generation Techniques: An Empirical Study on GitHub Copilot (2023)[4]

Authors: Antonio Mastropaolo, Luca Pascarella, Emanuela Guglielmi, Matteo Ciniselli, Simone Scalabrino, Rocco Oliveto, and Gabriele Bavota

- Study focused on evaluating the robustness of GitHub Copilot for code generation.
- 892 Java methods generated using original and paraphrased natural language descriptions.
- Two paraphrasing techniques used:
  - PEGASUS (deep learning-based)
  - Translation Pivoting (TP)

Tools :

- GitHub Copilot
- Java (for code generation)
- PEGASUS (NLP model), TP pipeline

LITERATURE SURVEY 4 (CONTD..)

Advantages:

- Demonstrated how input phrasing significantly affects code quality.
- Useful for improving design of code recommendation tools.
- Explores automation in paraphrasing for testing code generation models.

Disadvantages:

- High variability in Copilot's output can reduce reliability.
- Paraphrasing may result in inconsistent or inaccurate code suggestions.

LITERATURE SURVEY 5

Leveraging pre-trained language models for code generation(2024)[5]

Author: Ahmed Soliman,Samir Shaheen,Mayada Hadhoud

- Model Selection:BERT, RoBERTa, ELECTRA, and LUKE.
- Used Marian as a decoder for enhanced code generation.
- Datasets: CoNaLa and DJANGO.
- Preprocess Data: Tokenize and normalize.
- Fine-Tuning: Train models on CoNaLa and DJANGO.

  Tools :

  - Programming Language: Python
  - Deep Learning Framework: PyTorch
  - HuggingFace Transformers,HuggingFace Trainer,Google Colab Pro
  - RAM: 16 GB minimum, 32 GB preferred.

LITERATURE SURVEY 5(CONTD..)

Advantages:

- Better code quality with improved BLEU scores and exact match rates.
- Speeds up coding and reduces manual effort.
- Models provide better contextual and knowledge integration.

Disadvantages:

- Small dataset size affects generalizability.
- Focus on single-line code generation.
- Only a subset of models tested, missing potential improvements.

LITERATURE SURVEY 6

Code Generation from Flowchart using Optical Character Recognition Large Language Model (2024)[6]
Author:Aryaman Darda and Reetu Jain

- Flowchart image is processed using OCR and deep learning to extract text, which is then sent to LLaMA 2-Chat to generate Python code with explanation.
- Gradio interface allows users to upload images and view the generated code easily.

Advantages:

- Helps beginners learn fast by turning flowchart pictures into reliable Python code with an easy-to-use screen.

Disadvantages:

- OCR may struggle with low-quality or handwritten flowcharts

LITERATURE SURVEY

Title	Year	Methodology	Advantage	Disadvantage
An Extensible Approach to Generate Flowcharts from Source Code[1]	2018	-Code to flowchart -Three stages	Allows independent updates	Knowledge on Compiler design,AST

LITERATURE SURVEY

Title	Year	Methodology	Advantage	Disadvantage
Code Similarity Detection Using AST and Textual Information [2]	2019	-Generate Simhash fingerprints -Extract AST and Compute edit distance -Weighted similarity calculation	Considers both text structure similarity.	No run check

LITERATURE SURVEY

Title	Year	Methodology	Advantage	Disadvantage
Automatic Code Generation for C and C++[3]	2021	Users draw flowcharts with predefined shapes in GUI, converted to C/C++ code.	-Easy to use GUI -Reduces syntax errors	Needs User interaction for each step

LITERATURE SURVEY

Title	Year	Methodology	Advantage	Disadvantage
On the Robustness of Code Generation Techniques: An Empirical Study on GitHub Copilot [4]	2023	-Generated 892 Java methods. -Tested with original and paraphrased inputs. -Used PEGASUS and Translation Pivoting.	-Shows impact of input phrasing. -Useful for code tool enhancement.	-Output inconsistency. -Sensitive to paraphrasing.

LITERATURE SURVEY

Title	Year	Methodology	Advantage	Disadvantage
Leveraging Pre-Trained Language Models for Code Generation[5]	2024	- BERT, RoBERTa, ELECTRA, LUKE. - Marian as decoder. - CoNaLa, DJANGO datasets. - Tokenize, normalize data. - Fine-tune models.	- Higher BLEU scores. - Faster coding. - Better contextual integration.	- Small datasets. - Single-line focus. - High computational needs. - Limited models tested.

LITERATURE SURVEY

Title	Year	Methodology	Advantage	Disadvantage
Code Generation from Flowchart using Optical Character Recognition and Large Language Model [6]	2024	OCR extracts text from flowchart, LLaMA 2 generates Python code	Beginner-friendly, focuses on logic	Performance issues with unclear images.

CONCLUSION FROM LITERATURE SURVEY

- The literature survey highlights key limitations in existing systems used in programming education and code analysis.
- Many systems require manual flowchart creation, making the development process slow and inefficient.
- The learning flow is often reversed, focusing on code before understanding the underlying algorithm and structure.
- Code similarity detection is mostly limited to plagiarism detection, lacking educational feedback or performance evaluation.
- This makes the need for an automated and user-friendly solution that makes code generation and assessment easier and more effective.

PRODUCT FUNCTIONS

- Accepts natural language prompts from the user and converts them into well-structured pseudocode for further processing.
- Uses rule-based mapping to transform the pseudocode into a clear algorithm and a corresponding visual flowchart.
- Automatically generates accurate Python code from the input prompt using a Huggingface model.
- Provides an integrated environment for running, testing the generated or user-submitted Python code.
- Evaluates the student's code by comparing it with the generated code, offering automated grading and meaningful feedback.
- Ensures smooth usage through a clean and user-friendly interface

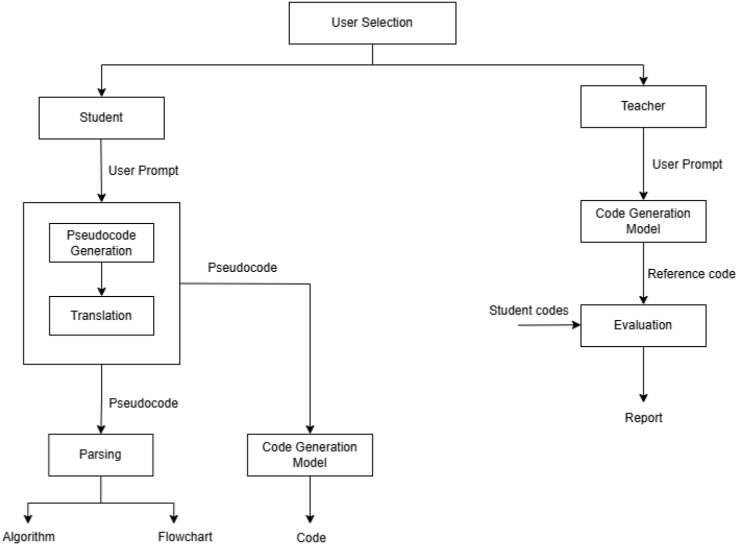
SOFTWARE REQUIREMENTS

- **Language**: Python.
- **Framework**: Flask
- **Frontend Technologies**: HTML, CSS ,Javascript
- **API**: Blackbox.ai
- **Flowchart Generation**: Graphviz
- **Code Generation Model**:HuggingFaceTB/SmolLM2-1.7B-Instruct
- **Development Environment**: Visual Studio Code.

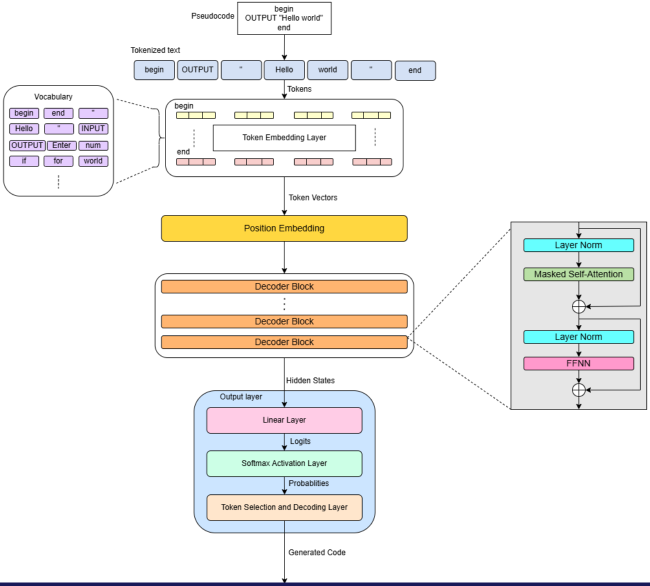
HARDWARE REQUIREMENTS

- **Operating System:** Windows 10/11 64-bit or equivalent
- **Minimum CPU:** Quad-core or higher (e.g., AMD Ryzen 5 / Intel i5 or better)
- **Minimum RAM:** 8 GB, Recommended 16 GB
- **Minimum GPU:** 2 GB
- **Minimum Storage:** At least 256 GB SSD

SYSTEM ARCHITECTURE

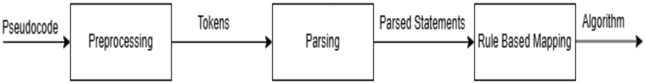


CODE GENERATION ARCHITECTURE

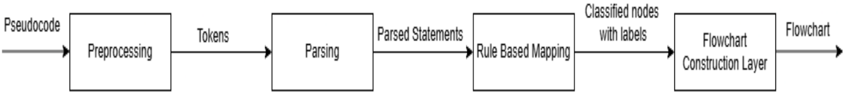


ALGORITHM,FLOWCHART GENERATION - ARCHITECTURE

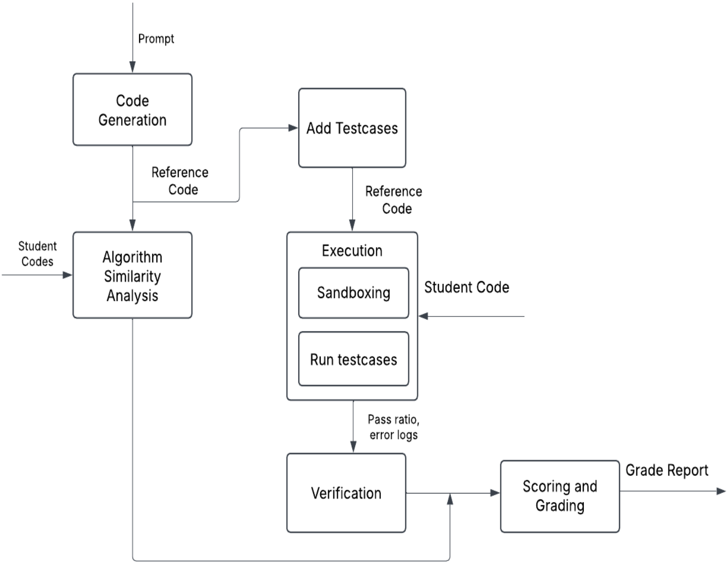
Algorithm Generation Architecture



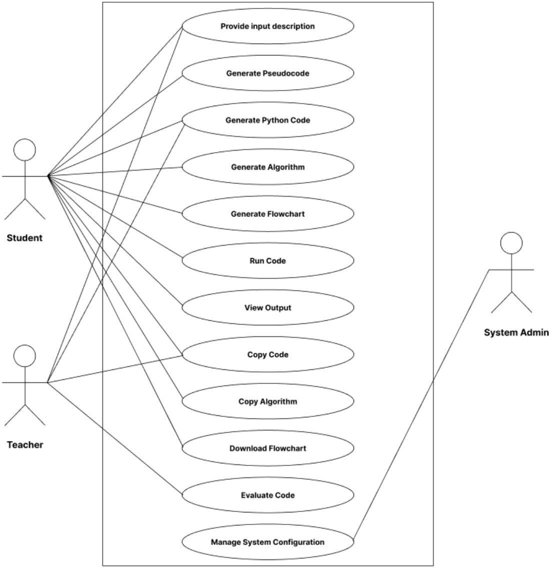
Flowchart Generation Architecture



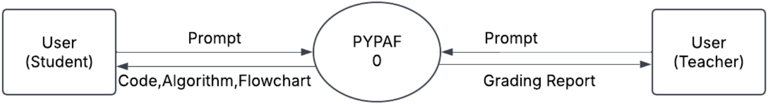
EVALUATION ARCHITECTURE



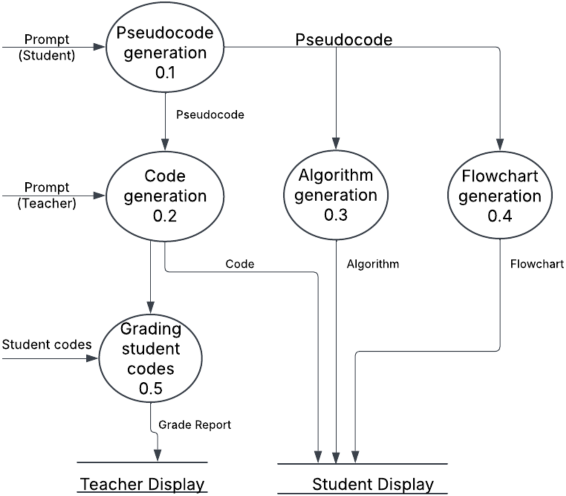
USECASE DIAGRAM



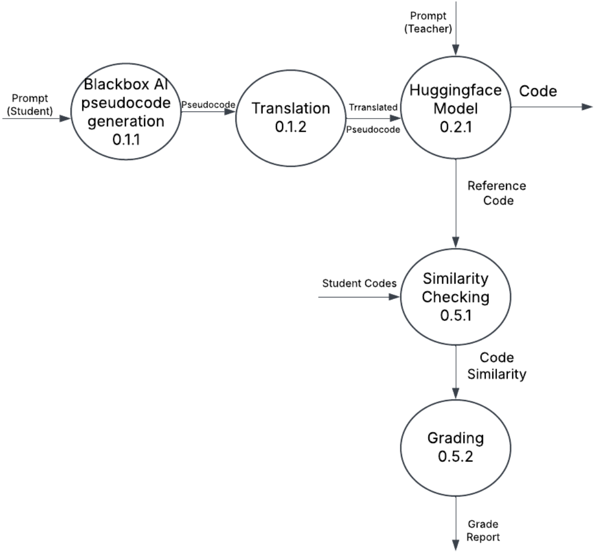
DFD LEVEL 0 DIAGRAM



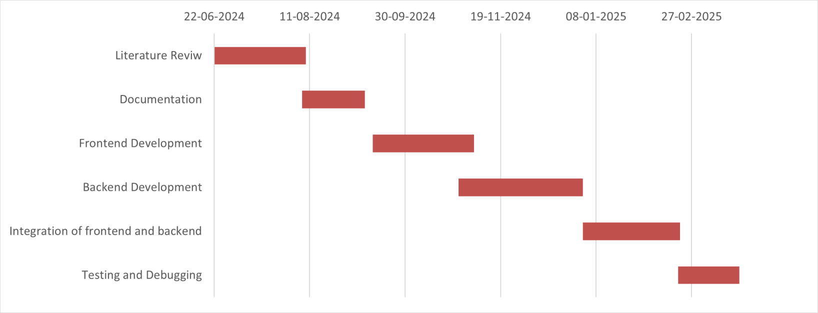
DFD LEVEL 1 DIAGRAM



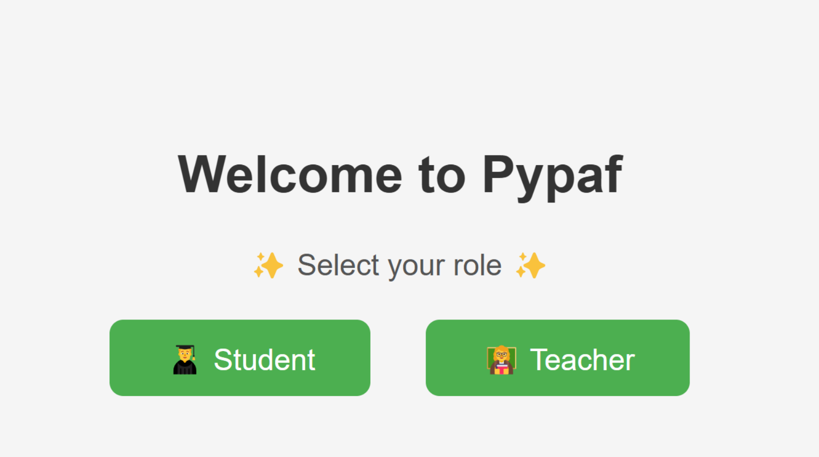
DFD LEVEL 2 DIAGRAM



GANTT CHART


22-06-2024
11-08-2024
30-09-2024
19-11-2024
08-01-2025
27-02-2025
Literature Reviw
Documentation
Frontend Development
Backend Development
Integration of frontend and backend
Testing and Debugging

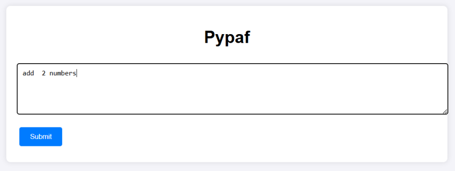
IMPLEMENTATION & RESULT



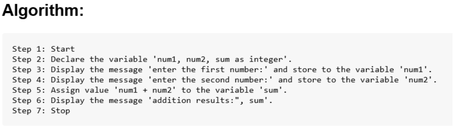
Welcome to Pypaf

✨ Select your role ✨

IMPLEMENTATION & RESULT



add 2 numbers

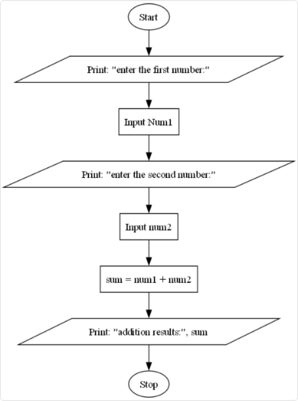


Algorithm:

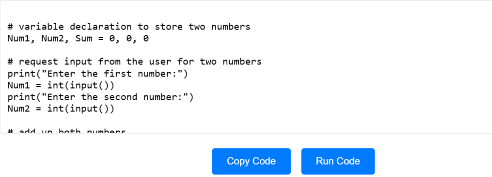
```
Step 1: Start
Step 2: Declare the variable 'num1, num2, sum as integer'.
Step 3: Display the message 'enter the first number:' and store to the variable 'num1'.
Step 4: Display the message 'enter the second number:' and store to the variable 'num2'.
Step 5: Assign value 'num1 + num2' to the variable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

IMPLEMENTATION & RESULT

Flowchart:


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

Generated Python Code:

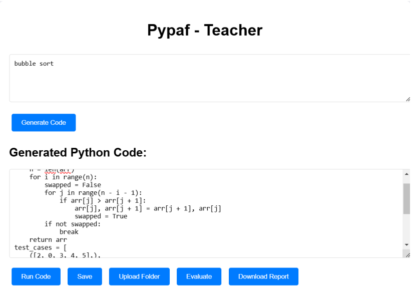


```
# variable declaration to store two numbers
Num1, Num2, Sum = 0, 0, 0

# 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
```

IMPLEMENTATION & RESULT

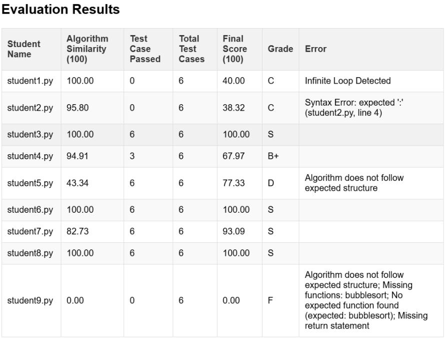


Pypaf - Teacher

```
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],),
```



Evaluation Results

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

CONCLUSION

- This project bridges the gap between theoretical learning and practical coding by guiding students from natural language to executable Python code.
- It improves programming education by presenting algorithms, flowcharts, and code in a structured and understandable flow.
- The tool encourages self-learning among students while also supporting teachers with automated code evaluation and feedback.
- Overall, it transforms the way students engage with programming by making the learning process more interactive, accessible, and effective.

REFERENCES

1. Damitha D. Karunarathna and Nasik Shafeek. (2018). *An Extensible Approach to Generate Flowcharts from Source Code*. International Journal of Research - Granthaalayah, 6(9), 505–519. https://doi.org/10.5281/zenodo.1465019
2. Wu Wen, Xiaobo Xue, Ya Li, Peng Gu, and Jianfeng Xu. (2019). *Code Similarity Detection using AST and Textual Information*. International Journal of Performability Engineering, 15(10), Article ID 2683. https://www.ijpe-online.com/EN/10.23940/ijpe.19.10.p14.26832691,
3. Sanika Patade, Pratiksha Patil, Ashwini Kamble, and Prof. Madhuri Patil. (2021). *Automatic Code Generation for C and C++ Programming*. International Research Journal of Engineering and Technology (IRJET), 8(5). e-ISSN: 2395-0056.

REFERENCES

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 Soliman, A., Shaheen, S., & Hadhoud, M. (2024). Leveraging pre-trained language models for code generation. *Complex Intelligent Systems*, 10, 3955–3980. `https://doi.org/10.1007/s40747-024-01373-8`

6 Darda, A., & Jain, R. (2024). Code generation from flowchart using optical character recognition & large language model. *Authorea Preprints*. `https://doi.org/10.36227/techrxiv.171392799.96378624`

Thank you