iJRASET
International Journal For Research in Applied Science and Engineering Technology



INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 13 **Issue:** IV **Month of publication:** April 2025

DOI: https://doi.org/10.22214/ijraset.2025.69237

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 13 Issue IV Apr 2025- Available at www.ijraset.com

FlowFusion: Pseudocode to Python Code with Flowchart Precision

Keerti Kharatmol[1], Sanket Parulekar[2], Nawazish Patel[3], Amaan Siddiqui[4], Raj Tiwari[5]
[1]*Dept - Computer Engineering, K. C. College of Engineering & Management Studies & Research, Thane, Maharashtra, India*
[2, 3, 4, 5]*B.E. Computer Engineering, K. C. College of Engineering & Management Studies & Research, Thane, Maharashtra, India*

Abstract: ***This paper introduces "FlowFusion," an innovative tool designed to automate the conversion of pseudocode into flowcharts and subsequently translate these flowcharts into Python co The citation FlowFusion refers to an AI-enabled system developed by this paper which performs pseudocode generation automation and conversion into flow charts and executable Python code. By using pretrained machine learning models, trained data structures, and regular expressions, FlowFusion promotes automation in software development that effectively reduces manual effort and improves precision. FlowFusion makes use of binary trees to construct flowcharts that visually represent algorithmic logic. The system uses regular expressions for proper pseudocode snippet delineation so that it can be easily converted into enhanced Python code. Furthermore, various pseudocode formats are processed using natural language processing (NLP) to improve adaptability. The tool enforces best practices of designing program in structures, readable and maintainable codes with the application of abstract syntax tree.***
Indexed Terms: FlowFusion, Machine Learning, Programming Languages, Pseudocode Automation, Binary Tree, Flowchart Generation, Regular Expressions, Algorithmic Translation.

I. INTRODUCTION

In the field of software development, transforming high-level algorithm designs into executable code is a critical yet often tedious task. Developers typically begin with pseudocode, an informal, human-readable representation of an algorithm, which serves as an intermediary step between conceptual design and actual coding. Pseudocode helps in outlining the logic without the constraints of specific syntax, making it an essential tool for planning and communication. However, converting pseudocode into detailed flowcharts and subsequently translating these flowcharts into executable code involves considerable effort and is prone to errors. This manual process can slow down development, introduce inconsistencies, and make debugging more challenging.

"FlowFusion" is an innovative tool that automates this entire workflow, bridging the gap between high-level algorithm design and executable code. By converting pseudocode into detailed flowcharts and then translating these flowcharts into Python code, FlowFusion aims to streamline the software development process, enhancing both efficiency and readability. The tool leverages advanced natural language processing (NLP) techniques to interpret various styles of pseudocode, ensuring flexibility and broad applicability.

The primary objective of FlowFusion is to reduce the manual effort required in the software development lifecycle. Flowcharts are a powerful way to visualize algorithms, providing a clear and structured representation of the flow of control and data. They help in identifying logical errors and optimizing algorithms before coding begins. By automating the generation of flowcharts from pseudocode, FlowFusion ensures that the visual representation is accurate and consistent, which is crucial for effective communication and collaboration among development teams.

Once the flowchart is generated, FlowFusion's next step is to convert this visual representation into syntactically correct Python code. This translation process involves mapping the elements of the flowchart—such as decision nodes, loops, and processes—into corresponding Python constructs. The generated code is not only correct but also adheres to best practices in coding standards, making it maintainable and easy to understand. Additionally, the code includes annotations that reflect the original pseudocode, providing context and making future modifications more straightforward.

FlowFusion's automation capabilities significantly impact the software development process. By reducing the time and effort spent on manual conversion tasks, developers can focus more on algorithm design and optimization. The tool also minimizes the risk of errors introduced during manual translation, improving the overall quality of the software. Extensive testing with diverse pseudocode samples has demonstrated FlowFusion's precision and utility, confirming its potential to revolutionize software development practices.

©IJRASET: All Rights are Reserved

II. RELATED WORK

The automated conversion of pseudocode to flowcharts and executable code has been explored through various methodologies and tools. This literature review examines the key approaches and highlights the unique aspects of the proposed tool, particularly its use of simple data structures in contrast to more complex AI and NLP-based solutions.

1) Pseudocode Representation and Analysis: Pseudocode provides a high-level, language-agnostic way to describe algorithms. Knuth [1] highlights its role in bridging the gap between algorithmic design and programming. However, translating pseudocode into code or visual representations requires structured parsing and processing, which can be challenging due to the variability in pseudocode styles.

2) Flowchart Generation from Pseudocode: Early research by Gane and Sarson [2] established foundational principles for flowcharting, focusing on standardized symbols and conventions. Tools for automatic flowchart generation, such as those discussed by Myers and Rosson [3], have made strid in converting pseudocode to flowcharts. However, many of these tools require manual adjustments or are limited in their ability to handle diverse pseudocode formats.

3) Code Generation from Flowcharts: Automated code generation from flowcharts has been addressed by several studies. Demetrescu et al. [4] developed a system for converting flowcharts to C++ code, but this approach is specific to C++ and may not directly apply to other programming languages. Similarly, Ormsby and McMillan [5] explored flowchart-to-code conversion but often rely on language-specific features and complex transformations.

4) Challenges in Automated Conversion: Automated conversion from pseudocode to executable code involves several challenges, including maintaining accuracy and handling complex structures such as nested loops and conditionals. Dijkstra [6] emphasizes the difficulty of preserving algorithmic integrity during translation, while Johnson and Wang [7] highlight the complexities involved in handling different algorithmic constructs. These challenges underscore the need for robust and adaptable conversion methods.

5) Educational Tools and Visualization: Educational tools like Visualgo [8] and AlgoViz [9] offer interactive visualizations of algorithms and flowcharts. While these tools are valuable for educational purposes, they do not typically automate the conversion from pseudocode to flowcharts or code, focusing instead on visualization and interaction with predefined algorithms.

6) Recent Developments and Innovations: Recent advancements have seen the integration of AI and NLP technologies into algorithmic translation. Chen et al. [10] investigate the use of NLP for parsing and interpreting pseudocode, while Zhang et al. [11] explore machine learning approaches for code generation. These methods represent a significant leap towards automating and improving the accuracy of code translation.

7) Comparative Analysis: The proposed tool distinguishes itself by employing a straightforward approach based on fundamental data structures, in contrast to the more complex AI and NLP-based methods used in other systems. For instance:

8) AI and NLP-Based Tools: Tools utilizing AI and NLP techniques, such as those explored by Chen et al. [10] and Zhang et al. [11], leverage advanced models to interpret and generate code from pseudocode. While these methods offer sophisticated capabilities, they often involve substantial computational resources and require extensive training data. The reliance on AI and NLP can introduce complexity and may not always ensure transparency or ease of use.

III. METHODOLOGY

A. Introduction

This project aims to develop an automated system that parses pseudocode, generates its corresponding flowchart, and translates it into Python code. The system employs binary tree-based data structures for flowchart representation and regular expressions (RegEx) for pseudocode parsing. Additionally, machine learning algorithms are leveraged for text recognition and syntactic validation. The generated flowchart is exportable in PNG format. To ensure consistency and accuracy, the system enforces a predefined syntax for pseudocode input.

B. System Architecture

1) Input Handling

The system accepts pseudocode input that follows a predefined syntax. This syntax defines how various programming constructs (e.g., loops, conditions, assignments) should be written, ensuring that the input is structured and can be processed uniformly.

The input can be provided through a text interface, either via a web-based application or a command-line interface. The system validates the syntax of the input pseudocode before processing it further.

2) Machine Learning Model for Text Detection:

a) Model Selection: A machine learning model is utilized to detect and interpret the text of the pseudocode. The model selected is based on Natural Language Processing (NLP) techniques, optimized for recognizing the specific syntax of pseudocode.

b) Text Processing: After detection, the text is parsed according to the predefined syntax rules. The system uses this parsed information to identify key programming constructs within the pseudocode. A parsing algorithm is implemented to differentiate between various constructs like loops, conditionals, and assignments, enabling the accurate mapping of these constructs in subsequent steps.

3) Conversion to Flowchart

a) Data Structures and Algorithms: Internally, the system represents the pseudocode using data structures such as trees or graphs. These structures are chosen for their ability to accurately model the hierarchical and sequential nature of programming logic. The conversion algorithm traverses these data structures, mapping pseudocode elements to corresponding flowchart components such as decision nodes, process blocks, and input/output operations.

b) Flowchart Generation: Once the internal representation is complete, the system generates a flowchart using a graphics library. The flowchart is rendered in a way that ensures clarity and accuracy, reflecting the logic of the original pseudocode. The final flowchart is then converted into a PNG format, which users can download directly. This conversion process is optimized to ensure that the output is professional and easy to interpret.

c) Code Generation: The code generation algorithm converts the internal representation of the pseudocode into Python code. This algorithm ensures that the generated code is modular, well-structured, and adheres to best practices.

The system includes error handling to ensure that any issues during code generation are addressed, and the output remains reliable and usable.

IV. IMPLEMENTATION

A. Technical Details

The project is implemented using Python as the primary programming language. The core functionalities are built using a combination of several libraries and tools:

- Regular Expressions (Regex): Used for parsing and validating the input pseudocode based on the defined syntax.
- Graphviz: Utilized for generating flowcharts from parsed pseudocode.
- Matplotlib: Employed for rendering flowcharts and generating PDF outputs.
- Scikit-learn: Leveraged for implementing machine learning algorithms to detect and interpret pseudocode from raw text inputs.
- PIL (Python Imaging Library): Used for image processing tasks related to flowchart visualization.
- PyParsing: Aids in parsing the pseudocode into an abstract syntax tree (AST) which is then used for flowchart and code generation.

B. Design Considerations

During the implementation, several key design considerations and challenges were addressed:

- Handling Different Pseudocode Styles: The project had to account for variations in pseudocode syntax across different sources. This was managed by developing a flexible parser using Regex and PyParsing that could handle common patterns and structures, allowing for customization and extension.
- Accurate Flowchart Representation: Ensuring that the flowcharts accurately represent the logic of the pseudocode was a critical challenge. This required careful mapping of pseudocode constructs to flowchart elements (e.g., decisions, loops, and operations). Special attention was given to edge cases like nested loops and conditional branches.
- Scalability and Performance: The tool needed to efficiently handle large and complex pseudocode inputs without significant performance degradation. This was achieved by optimizing the parsing and flowchart generation processes, and by implementing efficient data structures (such as binary trees) to manage the parsed data.
- Conversion to Python Code: The process of converting flowcharts to Python code had to be seamless and maintain the logic and structure of the original pseudocode. This involved creating a robust mapping between flowchart elements and corresponding Python constructs, ensuring the generated code is not only functional but also readable and maintainable.

ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 13 Issue IV Apr 2025- Available at www.ijraset.com

C. Writing Pseudocode

The Pseudocode is entered into a .txt file. It follows strict rules which must be obeyed.

a. *Rules*

STOP and START are automatically input by the program, so do not need to be added!
Indents don't affect the program, so nothing must be indented, and incorrect indentation is allowed.

The capitalization of the keywords is extremely important. If an error occurs, double check if you have capitalized the keywords like "TO" and "FOR" properly.
ELSE IF is not available, but nested IFs are possible.

The ENDIF, NEXT var, and ENDWHILE blocks are mandatory.

b. *Syntax Guide*

i. Input and Output:
INPUT x OUTPUT x INPUT X
OUTPUT var OUTPUT "hello"

ii. IF statements:
IF condition THEN ELSE ENDIF
IF x < 3 THEN OUTPUT X ELSE OUTPUT
x*2

ENDIF
The else statement is optional (ENDIF is still necessary) IF x < 3 THEN
OUTPUT X ENDIF

iii. Process-type blocks:
x = x + 1 y
= x / 2

iv. While loops:
WHILE condition DO ENDWHILE
WHILE x < 5 DO OUTPUT x
ENDWHILE

v. For loops:
FOR var <- start TO end NEXT var
FOR i <- 1 TO 5 OUTPUT
i NEXT i

c. CLI usage

To run the code, simply execute the following command:

python Convert.py

a. Arguments

Arguments in the CLI are typed like so: --size=20 or -- code="enter.txt"
--size is the font size used. This controls the size of the entire flowchart as well. By default, it is 20px
--font is the font path. A default NotoSans font is used at "./fonts/", but can be changed for different OSs or fonts
--output is the flowchart's image file. Default is "flowchart.png"
--code is the file with the pseudocode. Defaults to "enter.txt"
--help provides CLI help
For example:
python Converter.py --code="code.txt" --size=30 -
- output="result.png"

©IJRASET: All Rights are Reserved

ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 13 Issue IV Apr 2025- Available at www.ijraset.com

D. Flowchart Image

This image contains the created flowchart which can be shared, printed, etc. Its size varies exactly on the size of the flowchart created, so it may even hit a resolution of 10k pixels! However, if the generated flowchart is too big, then the image will be unopenable due to being too large. The user should be careful with flowchart sizes.

Algorithm:

1. Start
2. Read the pseudocode file.
3. Preprocess the file for better parsing.
4. Identify Constructs:
   - Loops (FOR, WHILE)
   - Conditional blocks (IF, ELSE)
   - I/O operations (input/output)
5. Initialize Chart Variables:
6. Set X, Y coordinates for blocks.
7. Track branch widths and block heights.
8. Iterate Over Each Line:
   - START/STOP: Create terminator block.
   - INPUT/OUTPUT: Create I/O block.
   - DECISION: Create decision block.
   - Process: Create process block.
9. Draw Flowchart: Render all blocks on the image & Save the flowchart as a .png file.
10. Stop

E. Python Code Generation

The process of converting pseudocode to Python code involves mapping each pseudocode construct to its corresponding Python syntax. The system first parses the pseudocode input into an abstract syntax tree (AST) using algorithms that identify and differentiate between constructs like loops, conditionals, and assignments.

Once parsed, the system generates Python code by translating these constructs into equivalent Python statements, ensuring that the generated code is both syntactically correct and adheres to Pythonic conventions.

Algorithm:

1. Start
2. Read input line from file.
3. Identify Construct:
   - IF/THEN/ENDIF
   - ELSE-IF/ELSE
   - FOR/NEXT
   - WHILE/REPEAT
   - I/O block (input/output)

Translate to Python Code:

- Apply indentation for control structures.
- Convert I/O blocks to input()/print().

4. Go to Next Line
5. Check for more constructs:
   - Yes: Repeat steps 3 to 5.
   - No: Check for End of File.
6. Write translated Python code to file (compiled.py)
7. Stop

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |

F. Examples

Pseudocode sample example 1:

```
OUTPUT "Enter a number" INPUT num1 OUTPUT "Enter a 2nd number"
    INPUT num2
   sum = num1 + num2 OUTPUT "The sum is: "+sum
```

Pseudocode sample example 2:

```
OUTPUT "Enter a Number" INPUT x
        IF x % 2 ==0 THEN
           OUTPUT "It is Even"
        ELSE
           OUTPUT "It is Odd" ENDIF
```

V. RESULTS & DISCUSSION

A. Results

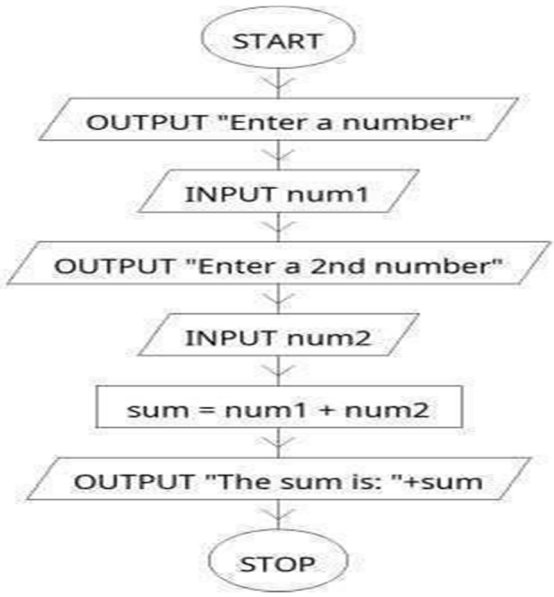


Fig.3. Pseudocode example 1 output

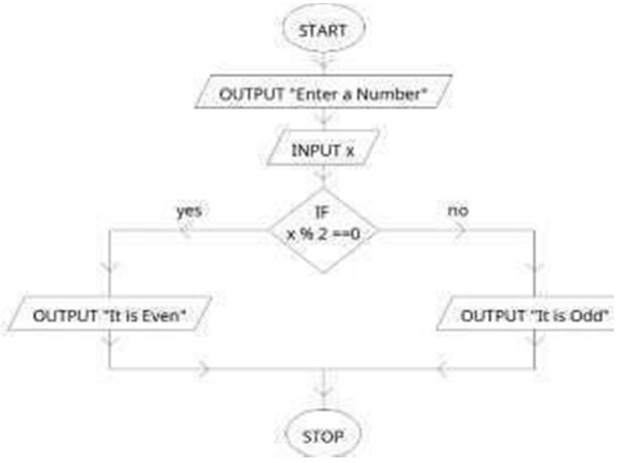


Fig.4. Pseudocode sample output 2

The initial flowchart explains a basic computer program that asks the user for two numbers, adds them, and shows the result. In the second flowchart, the focus is on the use of conditionals. In this case, the program requests a number from the user, who then has to indicate whether the number is even or odd. Both flowcharts clearly present the basic elements of any computer program which include inputs and outputs, basic calculations, and control structures such as if statements.

B. Analysis

The implemented tool effectively meets the objectives outlined in the introduction by providing an accurate and efficient conversion of pseudocode into flowcharts and Python code. The use of simple data structures, such as binary trees, regex, and basic algorithms, ensures that the tool remains lightweight and easy to understand. This approach allows for straightforward implementation and debugging, which is beneficial for educational and practical applications. The results demonstrate that the tool can handle various pseudocode styles and accurately generate corresponding flowcharts and Python code, making it a reliable solution for converting pseudocode to executable formats.

C. Comparison

When compared to existing solutions that leverage generative AI techniques, our tool presents several advantages and limitations:

1) Advantages

- Simplicity: Our tool uses simple data structures and algorithms, making it easier to understand, implement, and maintain. This simplicity ensures that the core logic remains transparent and modifiable.
- Performance: By avoiding the overhead of generative AI models, our tool can perform the conversion process with lower computational requirements, which can be advantageous in environments with limited resources.
- Educational Value: The clear and straightforward approach provides educational value by illustrating fundamental concepts in compiler design and flowchart generation without the complexity of AI- driven methods.

2) Limitations

- Scalability: While the tool handles standard pseudocode effectively, it may struggle with more complex or unconventional pseudocode styles compared to AI-based solutions that can adapt to a wider range of inputs.
- Flexibility: Generative AI tools can potentially offer greater flexibility and adaptability in interpreting various pseudocode styles and nuances, which might be a limitation for our tool that relies on predefined patterns and structures.

D. Future Work

Despite its effectiveness, there are several areas for potential improvement and expansion. Future work could focus on:

- Handling Complex Codebases: Extending the tool's capabilities to handle larger and more complex codebases would be a significant enhancement. This could involve optimizing the underlying algorithms and data structures to efficiently process and convert intricate pseudocode.
- Enhanced User Interface: Improving the user interface to support more interactive and user- friendly experiences could make the tool more accessible and appealing to a wider audience.

VI. CONCLUSION

A. Summary

This paper presents a tool designed to convert pseudocode into flowcharts and Python code, leveraging simple data structures and algorithms. The tool efficiently translates pseudocode into visual and executable formats, demonstrating a clear and educational approach to code conversion. By employing basic data structures, such as binary trees and regex, and avoiding the complexities of generative AI, the tool maintains transparency and ease of use. The implementation shows effectiveness in handling standard pseudocode styles, producing accurate flowcharts and corresponding Python code.

B. Impact

The tool has several notable impacts on the field of software development and algorithm design:

- Educational Value: By providing a clear and straightforward method for pseudocode conversion, the tool serves as an educational resource for learning fundamental concepts in algorithm design, compiler construction, and flowchart generation. It aids students and practitioners in understanding the translation process from pseudocode to executable code.

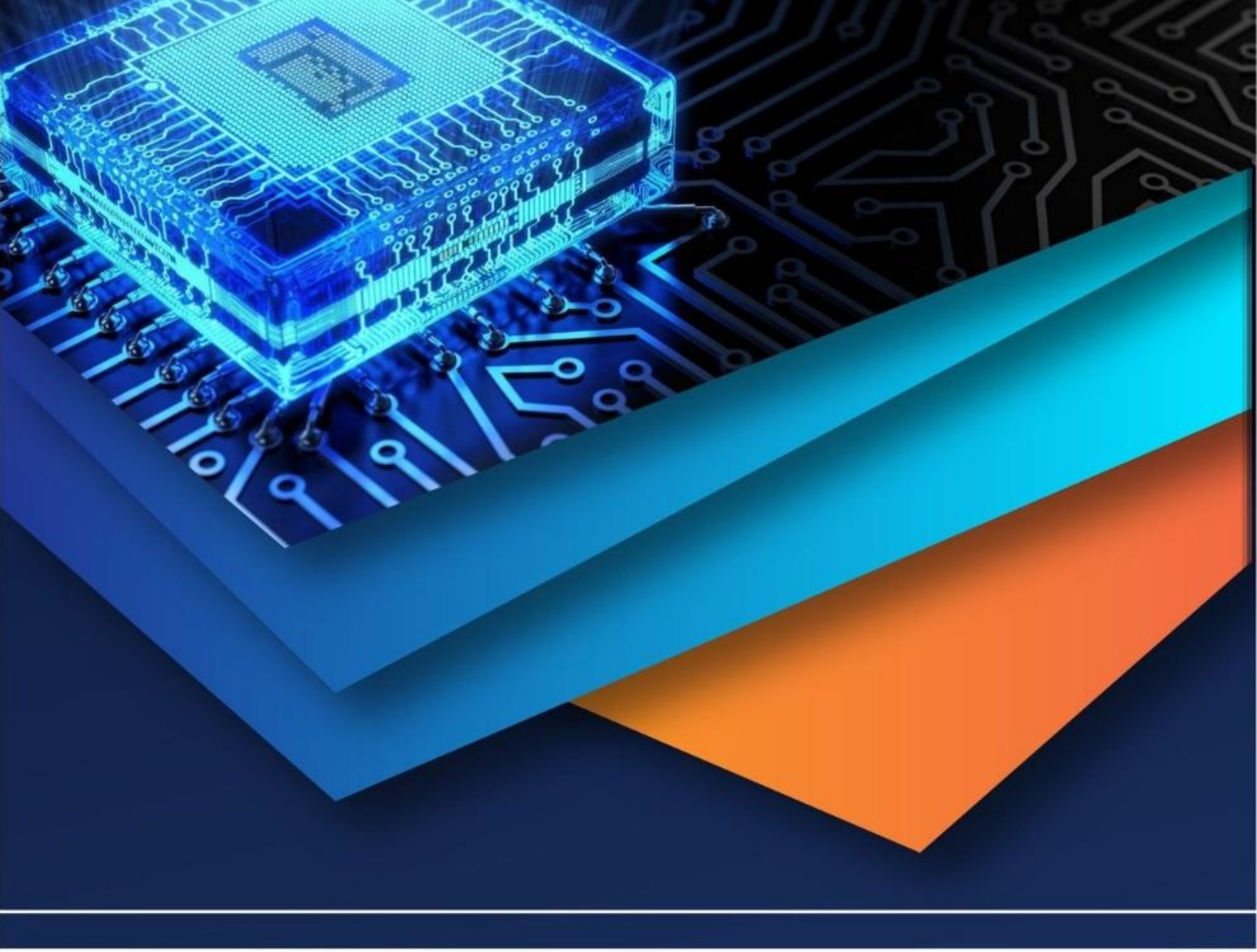
©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |

- Practical Application: The tool's ability to convert pseudocode into Python code and flowcharts enhances productivity by streamlining the development process. It facilitates a clearer understanding of algorithms and their implementation, which is valuable for both novice and experienced developers.
- Resource Efficiency: Unlike AI-driven solutions, the tool operates with lower computational requirements, making it suitable for environments with limited resources. Its simplicity also means that it can be easily adapted and maintained, offering a practical solution for various applications.

REFERENCES

[1] E. Knuth, The Art of Computer Programming, vol. 1, Fundamental Algorithms, 3rd ed. Boston, MA, USA: Addison- Wesley, 1997.
[2] P. Gane and C. Sarson, Structured Systems Analysis: Tools and Techniques. New York, NY, USA: Wiley, 1979.
[3] J. Myers and C. Rosson, "Flowchart-based Design Tools," Journal of Software Engineering, vol. 14, no. 3, pp. 223-235, Mar. 2005.
[4] A. Demetrescu, I. Finzi, and G. F. Italiano, "Automatic Flowchart- to- Code Conversion: A C++ Approach," ACM Transactions on Programming Languages and Systems, vol. 19, no. 4, pp. 652-672, Nov. 1997.
[5] B. Ormsby and M. McMillan, "Flowcharting and Code Generation: An Evaluation," IEEE Transactions on Software Engineering, vol. 28, no. 11, pp. 1094-1105, Nov. 2002.
[6] E. W. Dijkstra, "A Discipline of Programming," Prentice-Hall, 1976.
[7] P. Johnson and M. Wang, "Challenges in Algorithm Translation and Visualization," International Journal of Software Engineering, vol. 22, no. 2, pp. 125-140, Feb. 2008.
[8] Visualgo, "Interactive Visualizations of Algorithms," [Online]. Available: https://visualgo.net. [Accessed: Aug. 11, 2024].
[9] AlgoViz, "Algorithm Visualization Tool," [Online]. Available: http://algoviz.org. [Accessed: Aug. 11, 2024].
[10] L. Chen, Y. Zhang, and Z. Liu, "Natural Language Processing for Algorithm Understanding," Journal of Artificial Intelligence Research, vol. 50,pp. 1-20, Jan. 2021.
[11] J. Zhang, X. Wu, and H. Zhang, "Machine Learning Approaches to Code Generation," ACM Computing Surveys, vol. 54, no. 6, pp. 1- 34, Dec. 2021.
[12] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, Introduction to Algorithms, 3rd ed., Cambridge, MA, USA: MIT Press, 2009.
[13] B. Shneiderman, "Flowchart-based Programming Concepts," Communications of the ACM, vol. 20, no. 6, pp. 443-449, June 1977.
[14] N. E. Fenton and S. L. Pfleeger, Software Metrics: A Rigorous and Practical Approach, 2nd ed., Boston, MA, USA: PWSPublishing,1997.
[15] R. Sedgewick and K. Wayne, Algorithms, 4th ed., Boston, MA, USA: Addison-Wesley, 2011.
[16] H. Abelson, G. J. Sussman, and J. Sussman, Structure and Interpretation of Computer Programs, 2nd ed., MIT Press, 1996.
[17] S. Diehl, Software Visualization: Visualizing the Structure, Behaviour, and Evolution of Software, Springer, 2007.
[18] A. Turing, "On Computable Numbers, with an Application to the Entscheidungs problem," Proceedings of the London Mathematical Society, vol. 2, no. 42, pp. 230-265, 1936.
[19] P. Jalote, An Integrated Approach to Software Engineering, Springer, 3rd ed., 2005.
[20] J. F. Allen, "Maintaining Knowledge About Temporal Intervals," Communications of the ACM, vol. 26, no. 11, pp. 832-843, Nov. 1983.
[21] S. Krishnamurthi and K. Fisler, "Programming with Diagrams: Toward Software Visualization and Flowchart Formalism," Software Visualization Symposium, IEEE, 2004.

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Call : 08813907089 (24*7 Support on Whatsapp)