ISSN: 2581-1274

Development of an Automated Pseudocode to Python Translator Application with Three-Stage Compilation Visualization for Learning Compilation Techniques

Iis Aisyah [a,1,*], Aulia Ikhsan [a,2], Agung Siswopranoto [a,3]

[a] *Universitas Pamulang, Jl. Puspitek, Buaran, Kec. Pamulang, Kota Tangerang Selatan, Banten 15310*
[1] *dosen02694@unpam.ac.id*; [2] dosen02691@unpam.ac.id; [3] dosen02690@unpam.ac.id*
** corresponding author*

ARTICLE INFO

Article history:
Published
September, 14 2026

Keywords:
Pseudocode
Python
Compilation Visualization
Flask
Programming Education

ABSTRACT

Advances in information technology necessitate more effective programming education, particularly for vocational and early-stage university students struggling with algorithmic logic and language structures. A primary challenge is the transition from conceptual pseudocode to actual program code, such as Python. Misunderstandings of control structures, data types, and logical flow frequently cause syntax and logic errors. This study develops a web-based, automated pseudocode-to-Python translator application integrated with a three-stage compilation visualization: tokenization, parsing, and code generation. Employing a Research and Development (R&D) prototyping model and the Flask framework, the system features pseudocode input, interactive compilation visualization, direct Python code execution, and conversion result downloads. The application recognizes common pseudocode structures (e.g., IF–ELSE, FOR, WHILE, FUNCTION, INPUT, PRINT, SET, and RETURN) and automatically handles contextual input data types. Functional evaluation involved testing dynamic pseudocode scenarios across varying complexity levels and input simulations. Results indicate that the system enhances learners' comprehension of fundamental programming concepts and compilation processes, reduces syntax errors, and supports interactive, self-directed learning. This work contributes to programming education by integrating automated translation with compilation process visualization.

I. Introduction

The rapid advancement of information and communication technology (ICT) has transformed educational methodologies, necessitating more interactive and adaptive learning approaches. In computer science education, programming is a fundamental competency, yet novice students frequently struggle to comprehend foundational concepts such as data types, control structures, and program logic flow [1]. To mitigate these difficulties, pseudocode is commonly employed to conceptualize algorithms without the constraints of specific programming language syntax [2]. Despite its flexibility and simplicity, transitioning from pseudocode to an actual programming language like Python remains challenging. Students often confuse conceptual representations with strict syntax rules, leading to runtime and logic errors. For example, novices frequently misapply the assignment operator (=) for comparison (==) or incorrectly translate loop structures [3].

Traditional instructional methods, which rely heavily on static modules and theoretical explanations, often fail to provide a deep understanding of internal program execution. The compilation process encompassing lexical analysis, parsing, and code generation is typically explained only theoretically. Consequently, students cannot observe how tokens, logical structures, and Python code are formed from pseudocode. This hinders conceptual understanding and debugging skills, particularly when dealing with complex code, leaving students without a visual mental model of how compilers operate. [4].

Visualizing these pre-execution stages significantly enhances students' comprehension of code translation and optimization. Previous studies demonstrate that program execution visualization fosters correct mental models, improves debugging skills, and increases student engagement [5]. The integration of visual media bridges the gap between theory and practice, clarifying abstract concepts. For instance, web-based tools like VisOpt, which visualizes the synthesis phase of compilation, have successfully increased students' self-efficacy in understanding code optimization. Furthermore, interactive learning media and variable simulations enhance computational thinking, strengthening procedural understanding of program execution [6].

Although three-stage compilation visualization holds significant pedagogical potential, existing tools possess several limitations. They often lack seamless integration among automated translation, interactive visualization, and dynamic code execution. Furthermore, visualization systems introduce technical complexities in managing inter-stage representations. Without a well-designed user interface, such tools can impose an excessive cognitive load on novice programmers. Therefore, the system design must carefully balance technical depth with user comprehensibility [7].

To address these limitations, this study develops an automated pseudocode-to-Python translator application equipped with a three-stage compilation visualization and dynamic input simulation. Built as a web-based platform, the system enables students to interactively observe the transformation process. Sequentially, the tokenization stage produces a list of tokens, the parsing stage constructs the hierarchical logic structure, and the code generation stage produces executable Python code. The application recognizes common algorithmic structures, including IF–ELSE, FOR, WHILE, FUNCTION, INPUT, PRINT, SET, and RETURN. By displaying these processes interactively and allowing users to dynamically test input values, the application develops systematic debugging skills and raises awareness of a compiler's internal workings [8].

The urgency of this research encompasses three key perspectives: [9].

1. Education and pedagogy [10]: It addresses the ineffectiveness of traditional instructional methods and the high incidence of syntax and logic errors among novice students.
2. Technology and innovation: It overcomes the limitations of existing learning tools by seamlessly integrating automated translation, compilation stage visualization, and input simulation [11].
3. Scientific contribution: It provides an innovative, adaptive, and web-based educational system that serves as a reference for future programming learning tools [12].

Ultimately, this system aims to deepen conceptual understanding, improve debugging capabilities, and better prepare students for practical software development.

II. Research Method

1. *Approach and Development Model*

This study employs a Research and Development (R&D) approach utilizing the Prototyping software development model. As suggested by this model is effective for developing user-centered, web-based educational applications. This approach enables an iterative process where developers and users interact repeatedly to refine the system based on real-world feedback from end-users.

The Prototyping model is deemed the most suitable because this research emphasizes the development of a compilation-based automated translator application that must undergo multiple trials, validations, and revisions to achieve optimal results in supporting the learning of compilation techniques.

2. *Stages of The Prototyping Development Model*

The system development stages in this study refer to the Prototyping software development method, which emphasizes an iterative process among design, evaluation, and system refinement based on user feedback. This method was selected because it aligns with the characteristics of educational applications that require iterative adjustments to ensure the developed system is truly effective, user-friendly, and meets the needs of end-users.

In the context of this study, the Prototyping method is applied to develop an automated pseudocode-to-Python translator application featuring a three-stage compilation visualization. Each stage is designed systematically to support the learning objectives of compilation techniques and allow for continuous evaluation throughout the development process. The stages of the Prototyping method adapted in this research are as follows:

1) Requirement Analysis
   The requirements analysis stage aims to comprehensively identify and formulate system requirements, encompassing both functional and non-functional requirements. In this stage, an analysis is conducted regarding the difficulties experienced by students in understanding the transition from pseudocode to the Python language, particularly concerning control structures, logic flow, and compilation stages. The results of this analysis serve as the foundation for determining the application's primary features and the necessary characteristics of the compilation visualization.
2) Prototype Design
   This stage focuses on designing the initial prototype as an early representation of the system. The activities conducted include designing the user interface and the flow of the three-stage compilation visualization, namely tokenization, parsing, and code generation. Furthermore, the basic logical structure for converting pseudocode to Python is formulated, utilizing the Flask framework as the foundation for developing the web-based application.
3) Evaluation and User Feedback
   The initially designed prototype is subsequently evaluated by involving users, specifically students and lecturers. The evaluation is conducted to assess aspects of usability, clarity of visualization, and the effectiveness of the interface design in supporting learning comprehension. Feedback is obtained through direct observation and interviews, which is then utilized as the basis for system refinement.
4) Prototype II Development and Revision
   Based on the evaluation results and user feedback, further development of the prototype is carried out. This stage encompasses interface improvements, refinement of the translation logic, and the addition of supporting features such as code execution simulation and a facility to download the converted code. Following the revisions, the system is re-tested to ensure that every function operates according to the predetermined requirements.
5) Implementation and Testing
   In the implementation stage, the refined application is comprehensively tested using various pseudocode examples with varying levels of complexity. Testing is conducted to assess system stability, the accuracy of the pseudocode-to-Python translation results, as well as the clarity and consistency of the three-stage compilation visualization. This stage ensures that the application can be utilized optimally as a learning medium.
6) Final Evaluation and Documentation
   The final stage aims to evaluate the application's effectiveness in supporting the learning of compilation techniques. The evaluation is carried out by reviewing the test results and user responses to the application's usage. The entire process and research outcomes are then systematically documented as a research report, which also serves as the basis for formulating recommendations for further application development in the future.

III. Result and Discussion

1. *Implementation Of The Automated Translator Application*

The automated translator application developed in this study is a web-based system built using the Python programming language with the Flask framework. The selection of Flask is based on its lightweight and flexible characteristics, as well as its support for the rapid development of interactive web applications, making it suitable for the prototyping-based Research and Development (R&D) approach that emphasizes continuous iteration and refinement. Through this framework, the application is capable of integrating the pseudocode translation process, the visualization of compilation stages, and Python code execution within a single, unified learning environment.

Functionally, the application serves as an automated translator from pseudocode to the Python programming language. The system receives pseudocode entered by the user through a web interface, then processes it sequentially to generate executable Python code. The translation process is not solely oriented toward the final code output; it also displays the internal stages that occur during conversion, allowing users to understand how pseudocode is represented and transformed into valid Python instructions. The recognized pseudocode structures encompass common commands in algorithm learning, such as IF–ELSE, FOR, WHILE, FUNCTION, INPUT, PRINT, SET, and RETURN.

The implementation of this application directly reflects the R&D approach with the Prototyping model, where the system is developed through multiple cycles of design, testing, and revision. In the initial stage, the prototype focused on the fundamental capability of translating pseudocode to Python. Subsequently, based on testing results and user feedback, the system was refined by adding features for three-stage compilation visualization, dynamic input simulation, direct code execution, and a facility to download the converted Python code. This iterative development pattern allows the application to evolve according to user needs and predetermined learning objectives.

The use of this environment supports an efficient development process and facilitates functional testing of the application across various pseudocode scenarios. Furthermore, the integration of a safe execution mechanism ensures that the translated Python code can be run without compromising the system. Beyond being a mere conversion tool, the developed application is designed as a learning medium for compilation techniques. The visualization of the tokenization, parsing, and code generation stages provides a concrete overview of the internal compilation process, which is generally abstract for novice students. Thus, the application not only assists students in obtaining correct Python code but also reinforces their conceptual understanding of how an algorithm is processed and translated by a computing system. This approach renders the automated translator application an educational tool that supports the learning of programming and compilation techniques interactively and meaningfully.

2. *Implementation of Three-Stage Compilation Visualization*

The three-stage compilation visualization is the core of the automated translator application developed in this study. The visualization is designed to help students understand the internal process of translating pseudocode into the Python language gradually and systematically. The visualized stages refer to the fundamental concepts of compilation—namely tokenization (lexical analysis), parsing (syntax analysis), and code generation—which are educationally adapted according to learning needs.

2.1 Tokenization Stage Visualization

The tokenization stage functions to break down pseudocode into fundamental units called tokens. In this stage, each line of pseudocode entered by the user is redisplayed as a token representation recognized by the system. Tokens encompass keywords (such as IF, FOR, WHILE), variables, operators, and logical expressions used within the pseudocode. The tokenization visualization is displayed immediately after the user presses the "Translate" (*Terjemahkan*) button, allowing students to see how each line of pseudocode is processed by the system before further analysis. This approach helps students understand that the compilation process does not occur instantaneously, but rather through the prior separation of fundamental elements. This stage is an adaptation of lexical analysis in a compiler, though simplified to be easily understood by beginners.

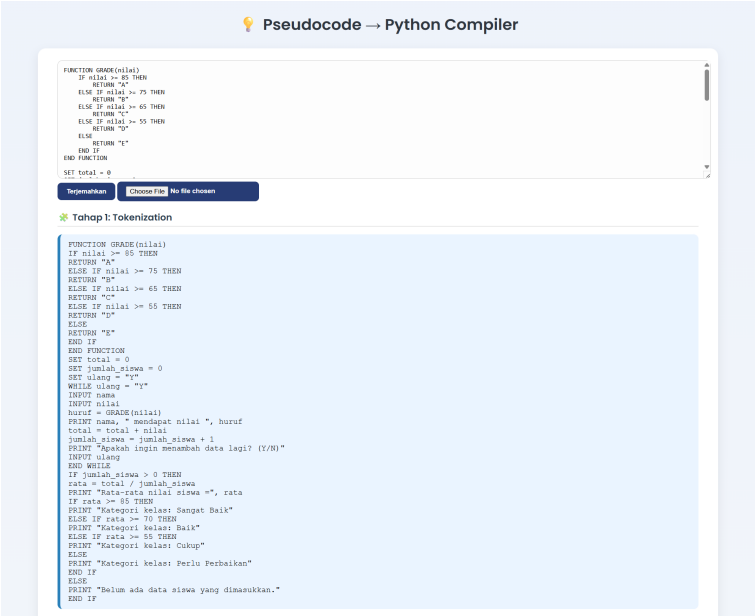

Fig. 1. Stage Tokenization

2.2 Parsing Stage Visualization (Syntax Structuring)

Once the tokens are recognized, the system proceeds to the parsing stage, which involves constructing the program's logical structure based on the generated tokens. At this stage, the application displays a hierarchical program structure that represents the relationships between logic blocks, such as IF–ELSE branching, FOR and WHILE loops, and FUNCTION definitions. The parsing visualization is presented in a nested structure, enabling students to easily follow the program's logic flow and understand the scope of each command block. By observing this structure, students can identify logical errors, such as the improper placement of conditions or mismatched control structures. This parsing stage refers to syntax analysis in compilation; however, it is focused on understanding the program flow rather than solely on formal syntax validation.

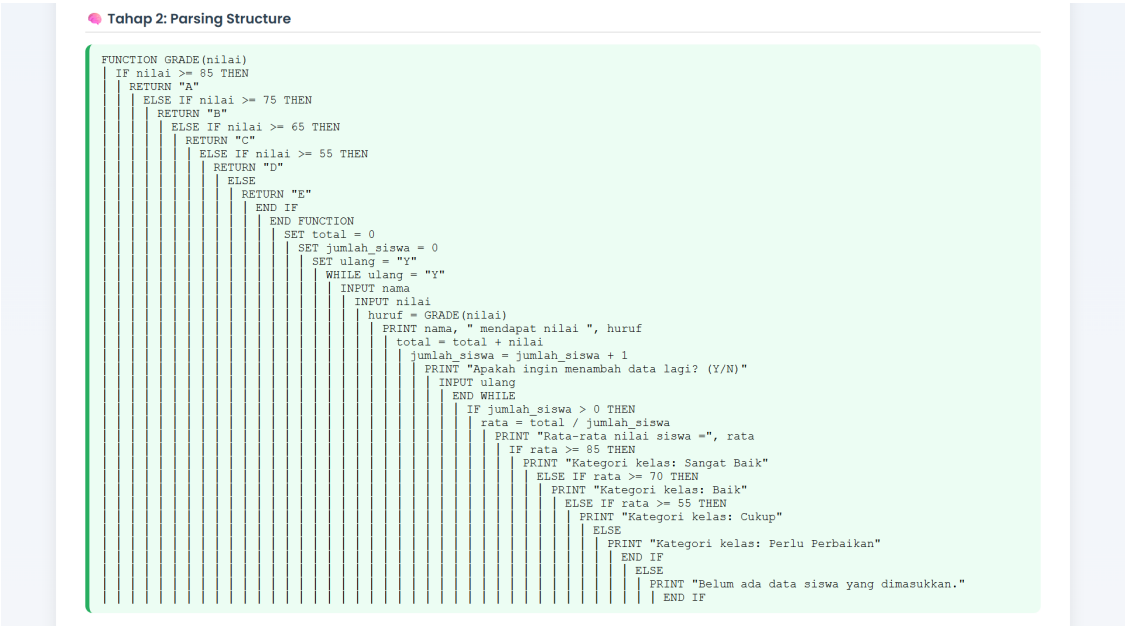

Fig. 2. Parsing Stage Visualization (Syntax Structuring)

2.3 Code Generation Stage Visualization

The final stage is code generation, which is the process of producing Python code based on the preceding parsing results. At this stage, the application displays the Python code that is automatically generated from the pseudocode entered by the user. The Python code is presented in a well-organized

and ready-to-execute format, complete with the appropriate indentation required by Python syntax rules.

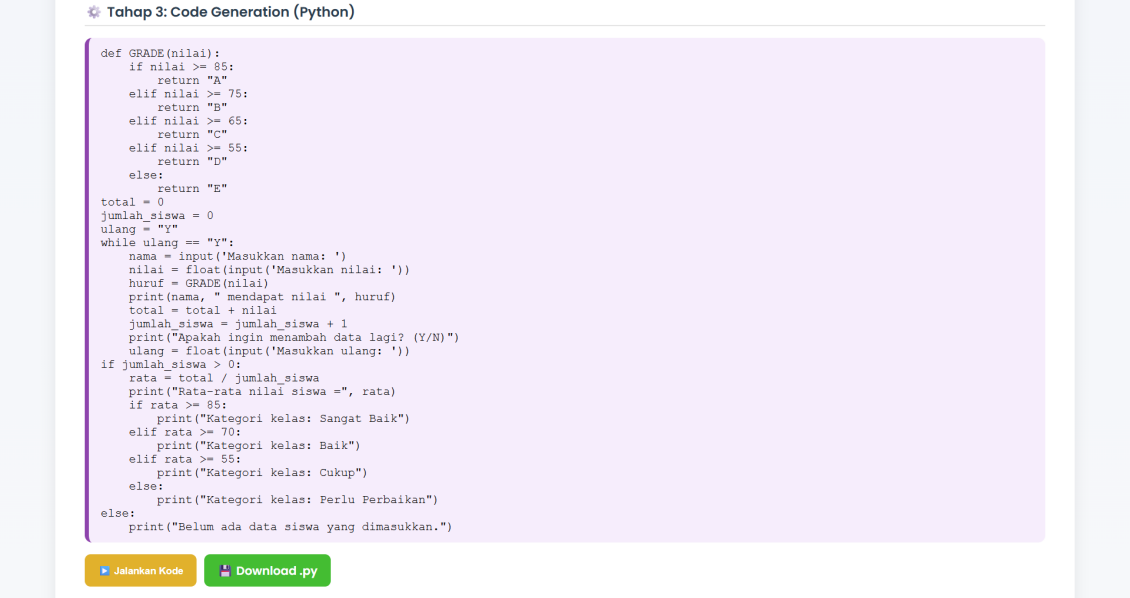

Fig. 3. Code Generation Stage (Python)

3. *Implementation of Simulation and Code Executing Features*

The input simulation and code execution features were developed to reinforce the application's role as an interactive learning medium, rather than merely a pseudocode-to-Python conversion tool. Through these features, users can execute the translated Python code directly within the application, while providing input values corresponding to the requirements of the tested algorithm.

During the implementation stage, the application allows students to enter input variable values that were previously defined in the pseudocode using the INPUT command. The system maps these input values to the generated Python code, enabling dynamic execution based on user-defined scenarios. This approach provides a more contextual learning experience, as students can observe the correlation among input values, computational processes, and the program's output.

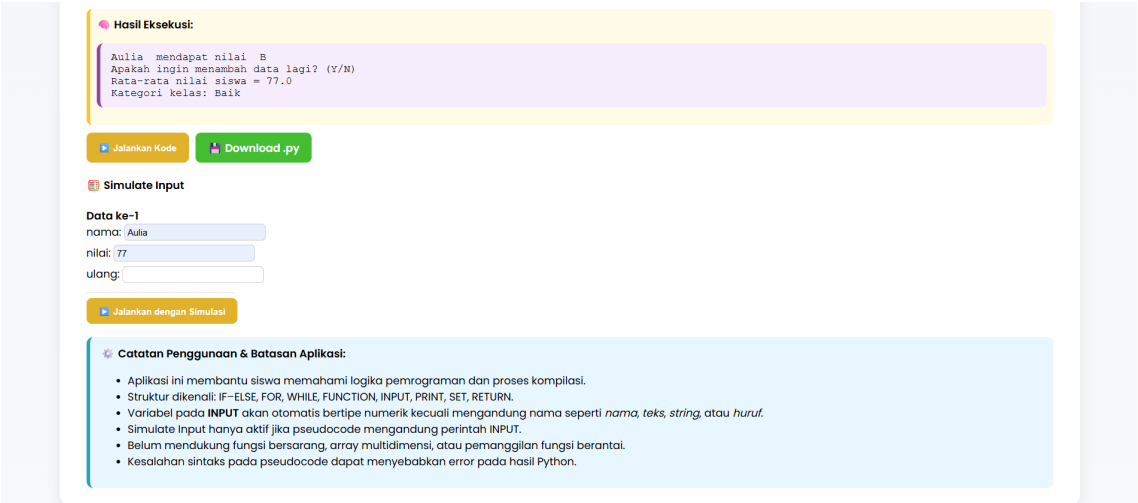

Fig. 4. Input Simulation of Pseudocode Translation Result

The code execution process is carried out in an integrated manner through the web-based application interface, without requiring additional configuration on the user's end. This allows algorithm simulation and testing activities to be conducted flexibly and efficiently within an educational context. With the input simulation and code execution features, users not only understand the translation process of pseudocode into Python code but can also verify the correctness of the algorithm's logic through the program's execution results. This feature supports an experiment-based learning approach, where students can try various input variations and directly observe their impact on the generated output.

4. *Application Functional Testing*

Functional testing was conducted to verify that all components and features within the automated translator application function according to the requirement specifications formulated during the requirements analysis stage. The primary focus of this testing encompasses the accuracy of the translation process from pseudocode to the Python language, the clarity and consistency of the three-stage compilation visualization, and the system's ability to stably handle variations in algorithmic structures. The testing process was conducted by applying several pseudocode scenarios that represent fundamental programming structures commonly used in learning, including:

1. Simple branching using IF-ELSE
2. Looping using FOR and WHILE
3. FUNCTION definition and invocation
4. The use of INPUT, PRINT, SET and RETURN
5. The combination of control structures within a single algorithm

The selection of these scenarios aims to test the system's flexibility in translating various forms of algorithmic logic. Each testing scenario was executed through systematic stages, starting from inputting the pseudocode into the application, observing the visualization results at the tokenization, parsing, and code generation stages, subsequently executing the translated Python code, and comparing the obtained output with the expected results based on the algorithm's logic.

The testing results indicate that the application consistently translates pseudocode to Python, generating executable code free of syntax errors. Furthermore, the visualization at each compilation stage is displayed sequentially and informatively, thereby facilitating users in understanding the flow of transformation from pseudocode to Python code. These findings indicate that the application has fulfilled the necessary functional aspects as an educational tool for learning compilation techniques.

5. *Evaluation of The Application's Role in Learning Compilation Techniques*

An evaluation of the automated translator application was conducted to assess its contribution in supporting the learning process of compilation techniques, particularly in assisting users to understand the fundamental concepts of program translation. The evaluated aspects include the users' level of conceptual understanding, engagement during the application's usage, and the ease of operating the available features. Based on observations during the application's usage, the three-stage compilation visualization provides a positive impact on users' understanding regarding the mechanisms of program translation. Users can directly observe how pseudocode is processed through the stages of lexical analysis, syntax analysis, and ultimately Python code generation, making the compilation process which was previously abstract more concrete and easier to comprehend.

Moreover, the presence of direct simulation and code execution features encourages users' active engagement in the learning process. Users are encouraged to try various algorithm variations and input values, as well as to observe the resulting changes in the output. This interaction contributes to an enhanced understanding of algorithmic logic and the structure of the Python programming language. Overall, the evaluation results demonstrate that this automated translator application possesses good effectiveness as a supplementary learning medium in compilation techniques courses or materials. The applied visual and interactive approach makes it relevant for use by vocational high school (SMK) students as well as first-year university students who require assistance in understanding programming concepts and the compilation process progressively.

IV. Conclusion

1. Conclusion

Based on the design, implementation, and testing results of the automated pseudocode-to-Python translator application with three-stage compilation visualization, it can be concluded that this research has successfully developed a web-based learning medium that supports the conceptual and practical understanding of compilation techniques. The application, developed using the Python Flask framework, is capable of translating pseudocode into valid Python code through the stages of tokenization, parsing, and code generation. The visualization of each compilation stage is presented sequentially and informatively, enabling users to understand that the program translation process does not occur instantaneously but rather through a structured mechanism.

The functional testing results indicate that the application can handle various fundamental algorithmic structures such as branching, looping, functions, as well as input and output with a high degree of accuracy and without generating syntax errors in the resulting Python code. Furthermore, the direct input simulation and code execution features provide an interactive learning experience and encourage users to experiment with algorithmic logic.

From an educational perspective, this application serves not only as a pseudocode-to-Python conversion tool but also as an educational medium that assists vocational high school (SMK) students and first-year university students in understanding the relationship among algorithms, syntactic structures, and the compilation process. Thus, the research objective of developing an automated translator application that supports the learning of compilation techniques has been achieved.

After the text edit has been completed, the paper is ready for the template. Duplicate the template file by using the Save As command, and use the naming convention prescribed by your conference for the name of your paper. In this newly created file, highlight all of the contents and import your prepared text file. You are now ready to style your paper; use the scroll down window on the left of the MS Word Formatting toolbar.

2. Recommendations

Based on the research that has been conducted, there are several recommendations to be considered for further development and future research:

1. Application development can be expanded by adding support for more complex pseudocode structures, such as arrays, data structures, recursion, and more detailed error handling, so that the system more closely approaches the characteristics of an actual compiler.
2. The visualization of the compilation process can be enhanced by incorporating graphical representations or animations of the program execution flow, allowing users to gain a more intuitive understanding of the relationships among the compilation stages.
3. The evaluation of learning effectiveness can be strengthened by involving a larger number of respondents and applying quantitative evaluation methods, such as pre-tests and post-tests, to measure improvements in user understanding more objectively.

This application has the potential to be developed into an independent learning platform integrated with practice modules, an automated grading system, and multi-programming language support, enabling its broader utilization within the context of secondary and higher education.

Acknowledgment

The authors wish to thank the Department of Informatics at Pamulang University for providing the resources and environment that facilitated the development of this application. Special thanks are also directed to all individuals, particularly the students and instructors, who provided valuable feedback during the prototyping and functional testing stages.

References

[1] A. Siswopranoto, G. Saputri, (2023). Penerapan Aplikasi E-Cuti Berbasis Web dengan Extreme Programming (Studi Kasus: Pengadilan Agama Jakarta Barat). Sainstech: Jurnal Penelitian dan Pengkajian Sains dan Teknologi. Vol. 33 No. 4, 70-74.

[2] A. Ikhsan et al., "SOSIALISASI GUNA PENINGKATAN KESADARAN MASYARAKAT," vol. 4, no. 1, 2025.

[3] C. K. Hasibuan, U. Islam, N. Sumatera, and P. Matematika, "Analisis Pembelajaran Algoritma Pemrograman Yahfizham Yahfizham manusia mampu menciptakan sebuah kemajuan teknologi yang

canggih dari zaman ke zaman . pemrograman , dimana algoritma pemrograman ini tidak hanya mencakup siswa atau," vol. 1, no. 5, 2023.

[4] I. Khairiah, "Algoritma Pemrograman : Studi Pustaka Pemahaman Algoritma Pemrograman," vol. 1, no. 4, 2023.

[5] A. Siswopranoto, S. Kom, M. Kom, Y. S. Kom, and M. Kom, KONVERSI PENGUNJUNG WEBSITE BERBASIS MACHINE LEARNING: Teori, Implementasi, dan Analisis. PENERBIT KBM INDONESIA, 2026.

[6] S. Rahayuningsih, Nurasrawati, and Muhammad Nurhusain, "Komparasi Efektivitas Model Pembelajaran Project Based Learning (PjBL) dan Konvensional: Studi Pada Siswa Menengah Pertama," Kogn. J. Ris. HOTS Pendidik. Mat., vol. 2, no. 2, pp. 118–129, 2022, doi: 10.51574/kognitif.v2i2.654.

[7] A. K. Syari, "SYSTEMATIC LITERATURE REVIEW : PENINGKATAN KEMAMPUAN," Algoritm. J. Math. Educ., vol. 6, no. 2, pp. 111–123, 2024.

[8] R. Koitz-hristov, F. Mandl, and F. Wotawa, "VisOpt – Visualization of Compiler Optimizations for Computer Science Education," pp. 603–609, 2025.

[9] R. Handayani, "EFEKTIVITAS PENGGUNAAN MEDIA VISUAL DALAM PEMBELAJARAN BAHASA INDONESIA DI SEKOLAH DASAR," JESS (Jurnal Sultra Elem. Sch., vol. 4, no. 2, 2023.

[10] G. Saputri and A. Siswopranoto, "Penerapan Aplikasi E-Perkara di Pengadilan Agama Jakarta Barat dengan Model Agile Development," Fakt. Exacta, vol. 18, no. 4, pp. 305–314, 2025.

[11] W. Zhang, Z. Wen, J. Pan, and W. Chen, "Visualization for Computer Program: A Survey," J. Comput. Des. Comput. Graph., vol. 35, pp. 1139–1149, 2024, doi: 10.3724/SP.J.1089.2023.19893.

[12] G. Saputri, A. Siswopranoto, and others, "Implementasi System Usability Scale Untuk Pengembangan Model Computer Based Test Pada Pesantren Nafidatunnajah," 2022.