Ai Powered Code Converter and Code Analyser – Code AI

Girish R,
Assistant Professor, CSE Department,
SNS College of Engineering, Coimbatore

Sheerin Farjana M,
IV Year CSE,
SNS College of Engineering, Coimbatore

Jeevitha S,
IV Year CSE,
SNS College of Engineering, Coimbatore

Surya B,
IV Year CSE,
SNS College of Engineering, Coimbatore

Krishna Kumar V,
IV Year CSE,
SNS College of Engineering, Coimbatore

Abstract:- Code developers and students often face challenges in translating code across various programming languages. An efficient code converter is vital to ensure accurate translations and minimize the time spent on manual rewriting. In today's fast-paced development environment, the ability to seamlessly transition code between languages enhances productivity and streamlines workflows, enabling developers to focus on higher-level tasks. However, many existing code converter tools fall short in key areas. A major issue is the production of inaccurate translations, which can introduce bugs and functional discrepancies that hinder development. Furthermore, these tools often lack customization options, limiting developers' ability to tailor conversions to specific needs. This lack of flexibility complicates debugging, as unclear representations of the original logic make it harder to identify and resolve issues. This project addresses these challenges with an AI powered code conversion tool designed to automate translation and analysis across multiple programming languages, including Java, Python, and C++. This innovative solution ensures functional equivalence by preserving the original code's performance and logic during the conversion process. It incorporates a code converter for automated syntax translation and a code analyzer that identifies vulnerabilities, verifies logical consistency, and suggests performance optimizations. By overcoming common pitfalls of existing converters, this tool enhances flexibility and usability, empowering developers and students to work confidently with diverse programming languages. With features like customizability and comprehensive analysis, the tool facilitates a more effective software development process, helping users harness the potential of different programming environments without compromising quality.

Keywords:- *AI-Powered Code Conversion, Code Analysis, Programming Language Translation, Automated Code Converter, Java, Python, C++, Functional Equivalence, Syntax Conversion, Code Optimization, Debugging, Logical Consistency, Software Development Productivity, and Customization.*

I. INTRODUCTION

As software development evolves, developers face challenges in maintaining code quality, enhancing efficiency, and ensuring compatibility across programming languages. With diverse languages, each with unique syntax and semantics, translating code while preserving functional integrity is complex. Manual methods, though effective at times, are labor-intensive, prone to human error, and cause inefficiencies. The need for a streamlined, error reducing solution is evident. Code AI addresses these challenges by automating code conversion and providing intelligent code analysis. Its core lies in a code converter that enables seamless translation between popular languages like Java, Python, and C++. Beyond simple language conversion, Code AI ensures functional equivalence, validating the code through iterative compilation and comparison to preserve performance and logic. The platform transcends basic translation by integrating a robust code analyzer that significantly enhances code quality and performance. It assesses translated code against best practices, focusing on three key areas: 1. Functionality Check ensures the code behaves as intended, aligning with design specifications and catching discrepancies early in the development cycle. 2. Loophole Detection identifies inefficiencies in loops and vulnerabilities like buffer overflows, enhancing both performance and security. 3. Optimization Suggestions improve runtime efficiency by analyzing algorithms, data structures, and control flow, recommending improvements to reduce resource consumption and enhance overall application performance. By automating labor- intensive tasks, Code AI saves time, mitigates human error, and enables developers to focus on higher-level tasks like architecture design. Its seamless integration into existing workflows enhances adoption and collaboration, accelerating development while reducing post- deployment bugs. Future advancements include machine learning for detecting complex issues, support for more languages, and real-time feedback during coding. With its integrated approach to code conversion, analysis, and optimization.

II. LITERATURE REVIEW

A. *Ms. Naziya Shaikh and Prof. Manisha Naik Gaonkar 2021, "Development of Intermediate Model for Source to Source Conversion ISOR Journal of Computer Engineering (IOSR JCE)" [1]*

The IOSR Journal of Computer Engineering (IOSR-JCE) launched a project in 2021 that automates source code conversion between programming languages using a template-based intermediate file approach. This system significantly reduces the time, cost, and errors associated with manual translations, facilitating software upgrades and code reuse between languages like Java and PHP. The project consists of two key components: Intermediate File Generation and Mapping to the Target Language. In the first step, the intermediate file captures the essential logic and structure of the source code while abstracting its syntax. By utilizing templates specific to the source language, this component ensures accurate representation and simplifies the subsequent conversion process. The second component converts the intermediate file into the target language using predefined templates that align with the target's syntax. This approach minimizes errors and maintains functional integrity by focusing on semantic equivalence rather than direct translation. Overall, the system aims to streamline code conversion, allowing developers to concentrate on coding logic and application design rather than tedious manual translations. By promoting code reuse and upgrades, this innovative solution enhances software development practices, benefiting developers and organizations alike.

B. *Eman J. Coco, Hadeel A. Osman and Niemah I. Osman (May 2018) "JPT : A Simple Java- Python Translator "Journal of An International Journal (CAIJ), Vol.5, No.2 [2]*

The JPT project (2018), developed by Eman J. Coco and colleagues, presents an innovative solution for translating Java code into Python, utilizing XML as an intermediary to streamline the code conversion process. This project focuses on facilitating the transition for developers by effectively handling basic programming constructs, making it easier for Java developers to adapt to Python's syntax and paradigms. By providing clear insights into the conversion process, the JPT project helps users understand not only how the code is transformed but also the rationale behind specific translation choices, thereby enhancing their learning experience. One of the standout features of the JPT project is its adaptable design. While its primary focus is on Java-to- Python translation, the architecture is flexible enough to accommodate future developments aimed at translating code between other programming languages. This adaptability addresses broader challenges in language interconversion, which is a critical aspect of software development in a multilanguage environment. As programming languages continue to evolve and diversify, tools like the JPT project become essential in bridging the gaps between languages, promoting code reuse, and easing the migration of legacy systems.

Ultimately, the JPT project contributes significantly to enhancing productivity and fostering collaboration among developers working across different programming languages.

C. *Christie Thottam and Imran Mirza (2024)"Intelligent Python Code Analyzer [IPCA]"Journal of (IJCRT) Volume 12, Issue 3 2024* [3]

Christie Thottam, Nigel Fernandes, Rehan Joseph, and Teacher Imran Mirza, is an progressed device outlined to improve Python improvement through AI driven and context- aware code investigation. By leveraging manufactured insights, the IPCA viably recognizes a wide extend of coding issues, from sentence structure blunders to potential execution bottlenecks, whereas too giving important optimization proposals that offer assistance designers progress the generally quality of their code. One of the standout highlights of the IPCA is its consistent integration with well known Python Coordinates Advancement Situations (IDEs), making it effortlessly open to designers in their commonplace coding situations. This integration upgrades the client involvement, permitting for real-time input as designers type in and alter their code. A key component of the IPCA is the Tree Traversal Calculation, which navigates the Unique Language structure Tree (AST) utilizing preorder, in-order, and post-order strategies. This intensive examination of the code structure empowers the analyzer to perform an in-depth appraisal of the code's rationale and semantics. By combining AI-driven bits of knowledge with progressed tree traversal methods, the IPCA altogether contributes to progressing coding hones, empowering designers to compose more effective, viable, and error-free Python code.

D. *Chongzhou Fang, Ning Miaoand others (2024)"Large Language Models for Code Analysis: Do LLMs Really Do Their Job?" Journal of international Journals, vol. 1, pp. 1-18, March 2024. [4]*

Changzhou Tooth, Ning Miao, and their group, speaks to a groundbreaking activity in the operation of expansive dialect models(LLMs) for improving law examination forms. This plan centers on building specialized datasets acclimatized for law examination, which are key for preparing LLMs to get it the subtleties of programming dialects and their isolated structures. The activity thoroughly assesses the capabilities of LLMs in insightful and assaying both standard and obscured law, which is especially appropriate in security- related assignments where law may be designedly clouded to evade revelation. By conducting nitty gritty case considers, the plan investigates how LLMs can be connected to genuine- world scripts, surveying their viability in relating vulnerabilities and certain inconveniences inside the law. To accomplish this, the plan utilizes progressed computerized devices, comparative as Theoretical Sentence structure Trees(AST), which oil a more profound understanding of the law's coherent influx and structure. By utilizing LLMs nearby modern investigation ways, the plan points to superior the delicacy and viability of law investigation, inevitably contributing to upgraded program security and strength in an steadily complex representation topography.

E. Gang Fan and their teams "Static Code Analysis in the AI Era: An In-depth Exploration of the Concept, Function, And Potential of Intelligent Code Analysis Agents " Journal of Ant Groups, China, 2023 [5]

Gang Fan and partners at Insect Gather, speaks to a noteworthy progression in bug disclosure and program quality confirmation by utilizing critical expansive dialect models LLMs) like GPT- 3 and GPT- 4. This inventive device addresses a basic challenge in computer program improvement the recurrence of wrong cons in bug reports. By working out the capabilities of LLMs, the ICAA has effectively diminished wrong cons from an scaring 85 to a more distant reasonable 66, whereas accomplishing an passionate review rate of 60.8. The examination handle facilitated by the ICAA is standard and user friendly. It starts with law accommodation, where the specialist surveys for violations and irregularities inside the codebase. Through progressed calculations and common dialect handling, the ICAA judgments issues successfully, outfitting formulators with a clearer understanding of understood issues. Once the investigation is total, the ICAA produces point by point reports that enlighten connected issues and offer practicable suggestions for enhancement. In spite of the tall privileged costs related with utilizing LLMs, the ICAA appears extraordinary promise in improving bug disclosure and determination forms. By streamlining law survey workflows, it points to meliorate program quality, decrease advancement time, and contribute to the creation of more distant vigorous and dependable operations.

F. Eric Jin and Yu Sun(2021) " An Calculation- Versatile Source law Engine to Computerize The summarizing From Python To Java"Diary in AIRCC Distributing Enterprise, USA,. [6]

Eric Jin and Yu Sun(2021) created an inventive device that computerizes the transformation of Python law into Java, accomplishing an enthusiastic delicacy rate with lower than 10 violations whereas preserving the unique usefulness. This plan is especially critical in moment's multi- dialect advancement landscape, as it streamlines cross language rendering for both fledglings and prepared experts. Working out an count approach, the apparatus successfully interprets different Python structures closely resembling as records, lexicons, and capacities into their Java rivals, subsequently facilitating the move for formulators who may be more commonplace with Python. To guarantee the tool's duty and adequacy, it was thoroughly approved and tried against a extend of USACO(USA Computing Olympiad) comes about and client- submitted law tests. This comprehensive testing handle not as it were illustrated the tool's capability to handle a diverse set of rendering scripts but too given perceptivity into its qualities and verifiable zones for advancement. stored nearby striking calculations like Google's GWT(which changes over Java to JavaScript) and Facebook's HipHop(which interprets PHP to C), this Python- toJava engine offers a down to earth and successful result for formulators looking to base the hole between dialects. By streamlining the transformation handle, the instrument improves efficiency and brings down the blockade to passage for those modern to Java, eventually contributing to a assist flexible and successful coding involvement in an progressively multilingual advancement landscape.

III. EXISTING FRAMEWORK

The existing framework of the Code AI AI- powered Code Converter and Analyzer is planned to automate the change of code between various programming dialects, counting Java, Python, and C++. It points to streamline the traditionally manual and error- prone handle of code interpretation, diminishing improvement time and minimizing human blunder. The framework works by analyzing the source code, changing over it into an halfway organize, and at that point creating the corresponding code in the target dialect whereas maintaining utilitarian keenness. Also, the Code Analyzer component of the framework performs a intensive assessment of the interpreted code, checking for best hones, effectiveness, and security. It gives recommendations for optimizing code execution, distinguishing bottlenecks, and identifying potential vulnerabilities. This guarantees that the changed over code not as it were capacities as anticipated but too meets tall guidelines of quality and security. The integration of both the code converter and analyzer offers a comprehensive arrangement that empowers designers to make strides their productivity, code quality, and by and large development productivity.

IV. PROPOSED APPROACH

Code AI is an AI-powered application arranged to revolutionize how originators related with source code by improving code change between programming tongues while improving code quality through examination and optimization. By utilizing made bits of knowledge machine learning, and common lingo processing (NLP), Code AI computerize time consuming assignments, lessens botches, updates code efficiency, and encourages moves between programming lingos. One of Code AI's center features is its capacity to alter over source code from one programming tongue to another. The change engine utilizes a template based midway record approach, abstracting the code's basis and putting absent it separately from linguistic unpretentious components. The system then maps this midway record to the target language, ensuring semantic proportionality instead of facilitate sentence structure elucidation. Predefined templates for both source and target tongues help minimize goofs. The engine ceaselessly improves its precision utilizing machine learning, training on perpetual datasets of code tests over multiple tongues and frameworks. Another key incorporate is the advanced AI-driven code analysis engine. This engine analyzes the quality of the source code, recognizing issues such as dialect structure botches, execution bottlenecks, security vulnerabilities, and best sharpen violations. By combining inert code examination, pattern affirmation, and significant learning, the engine gives vital bits of information into potential optimizations and upgrades. Facilitates into well known IDEs, it offers real-time input as architects compose code, highlighting districts for improvement, prescribing optimizations, and identifying potential security flaws.

The NLP component additionally offers human-readable explanations for recognized issues, making a distinction developers get it why certain coding practices may be unsafe. Code AI not as it were identifies issues but as well gives recommendations for courses of action. The optimization engine prescribes ways to update performance by murdering dreary operations, unraveling complex capacities, or implementing more compelling calculations. Machine learning is associated to analyze designs in coding and recognize common execution issues, allowing the system to propose specific optimization strategies. Moreover, Code AI encourages industry best sharpens through its refactoring capabilities, promoting proposition to improve coherence, common sense, and scalability. For case, it might recommend replacing significantly settled circles with more efficient calculations or utilizing more suitable data structures. Botch area and exploring are critical components of Code AI. The AI illustrate analyzes the code for potential runtime and logical botches that appear lead to crashes or unexpected behavior. By leveraging known bug patterns and machine learning desires, the system recognizes unsafe zones in the code before execution. The examining system makes a difference developers by prescribing fixes, publicizing code snippets, and coordinating them through complex debugging scenarios, engaging speedier identification and assurance of issues. Security is a key concern in cutting edge computer program development, and Code AI planning advanced security examination highlights. The AI appear channels the code for common vulnerabilities like SQL injection, cross-site scripting (XSS), and buffer overflows. By analyzing code plans and using chronicled data of known vulnerabilities, the system recognizes potential security perils. Code AI gives critical bits of information on securing the code, endorsing best practices, secure libraries, and frameworks, while disturbing engineers to untrustworthy sharpens and promoting course on how to address vulnerabilities. This ensures engineers make secure applications and decrease the likelihood of security breaches. The organize highlights an intuitive client interface (UI) that planning seamlessly into common change workflows. Open through a web-based platform, browser developments, and IDE plugins, it ensures architects can successfully get to the device across their favoured coding circumstances. Users can input code, select target tongues, and begin examination or change with irrelevant effort. The UI as well supports multi-language functionality, openness gadgets, and flexible compatibility, making it usable for diverse populations, checking people with failures. To help bolt in clients, Code AI offers educational resources such as webinars and tips to offer help originators make strides their coding practices. AI-based Code Converter and Analyzer system streamlines program development by promoting creators a competent tool for changing over, analysing, and optimizing code. By leveraging advanced AI and machine learning procedures, Code AI makes coding more viable, error-free, and secure. It makes a contrast improve code quality, decrease change time, enhance program security, and progress way better development sharpens, inevitably changing the way code is composed, kept up, and executed.

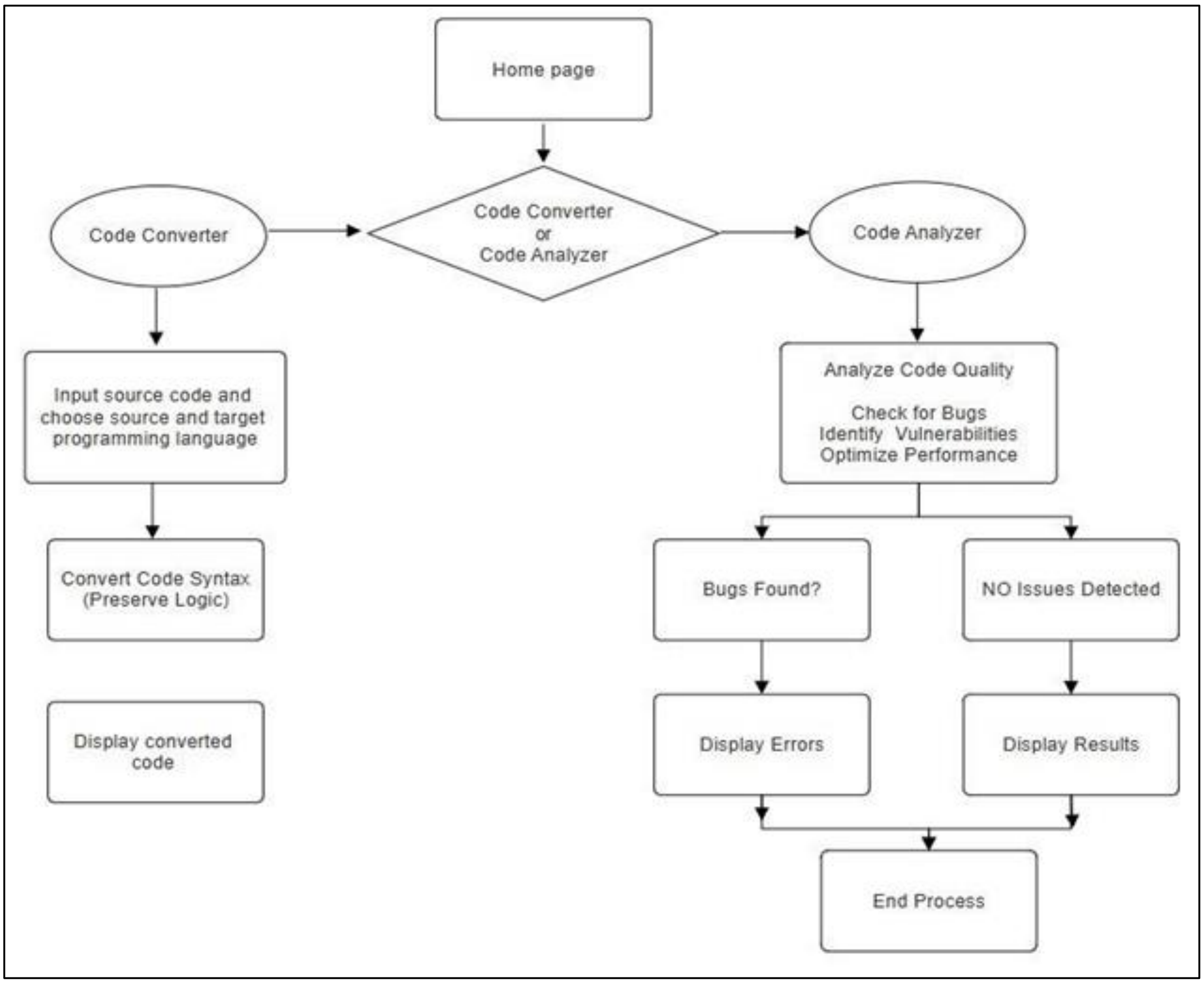

Fig: 1 Flow Diagram

V. CODE AI - OVERVIEW

A. List of Modules and Its Working:

List of maintenance to ensure ongoing functionality, stability, and reliability for each module. Maintenance focus on verifying that features continue to work as expected after updates, optimizations, or changes to the application's environment

➢ *Home*

- Ensure all sections and navigation links load properly after UI or frontend updates.
- Verify links and buttons function correctly after routing changes.

➢ *Code Analyzer*

- Confirm code analysis runs smoothly with accurate results after backend or algorithm updates.
- Verify performance metrics display correctly post-engine updates.

➢ *Functionality*

- Check core functions perform consistently across devices after updates.
- Ensure compatibility with different browsers.

➢ *Loop Holes*

- Verify detection of inefficiencies and loop optimizations after analysis engine updates.

➢ *Optimization*

- Confirm accurate optimization suggestions after algorithm changes.
- Check display accuracy of suggestions post-backend updates.

➢ *Choose File*

- Test file selection and upload compatibility with various types and sizes after updates.

➢ *Load to Analyze*

- Ensure files load into the analyzer correctly and display accurate data post-update.

➢ *Code Converter*

- Verify code conversion remains accurate and error-free after engine updates.

➢ *Choose Language*

- Confirm smooth language selection and display accuracy after language module updates.

➢ *Upload*

- Test upload consistency and compatibility with supported file types after backend changes.

➢ *Choose Language – User's Preferred Language*

- Ensure language preference applies correctly across sessions after backend or frontend changes.

➢ *Specification*

- Check accurate display of language specifications after UI or content updates.

➢ *Convert*

- Confirm code conversion is error-free and functional post-algorithm or syntax updates.

This checklist supports consistent performance and user experience for Code AI's Code Converter and Analyser.

VI. RESULT

The execution of a code converter and analyzer application offers noteworthy benefits to the program advancement prepare. By computerizing code interpretation between programming dialects, the application boosts productivity, empowering engineers to center on higher-level errands like plan and include advancement. This is crucial in today's fast- paced situations, where dexterity is key. The application moreover upgrades exactness by minimizing blunders common in manual interpretations, such as sentence structure blunders and consistent irregularities. Its vigorous calculations guarantee that changed over code capacities accurately from the begin, diminishing investigating time and moving forward unwavering quality. Customization and adaptability are center highlights, permitting designers to alter the transformation prepare to meet particular venture needs. The coordinates code analyzer gives experiences into code quality, distinguishing vulnerabilities and proposing optimizations to keep up security and execution. Supporting numerous programming dialects, the application cultivates collaboration among assorted groups and advances a more effective advancement prepare. Its user- friendly interface makes effective highlights open to both amateur and experienced engineers without a soak learning bend. Security is too prioritized, guaranteeing restrictive code remains ensured amid change and examination, which builds client certainty and complies with industry guidelines. Generally, the code converter and analyzer streamlines the coding handle and improves collaboration, making it an basic instrument for engineers.

VII. CONCLUSION AND FUTURE WORK

Our Code Converter and Code Analyzer venture is outlined to address key challenges confronted by developers, understudies, and teachers in computer program development. A major challenge is changing over code between dialects like Java, Python, and C++. By robotizing this handle, we make strides efficiency and decrease blunders, guaranteeing functional precision in the changed over code. This automation permits clients to center on coding logic and usefulness, or maybe than manual translation. The coordinates Code Analyzer evaluates code quality, recognizing vulnerabilities and recommending optimizations to help clients create vigorous, high-quality software. It catches potential issues early, maintaining security and execution standards. Existing apparatuses frequently endure from inefficiency, moderate execution, and restricted language back. Our arrangement leverages AI to overcome these confinements, advertising a consistent, flexible client involvement. The shrewdly algorithms streamline change and give deeper experiences amid code examination. Moving forward, we arrange to grow dialect back and coordinated machine learning for more sophisticated examination. Eventually, our venture points to disentangle code transformation, allowing clients to center on development and creativity. By progressing efficiency through automation and shrewdly investigation, we trust to foster a collaborative coding environment where engineers, understudies, and teachers can thrive.

REFERENCES

[1]. P. M. N. G. Ms. Naziya Shakh, "Development of intermediate Model for Source to Source Conversion," IOSR Journal of Computer Engineering IOSR - JCE, pp. 1-5, 2021.
[2]. H. A. O. a. N. I. O. Eman J. Coco, "JPT : A Simple Java-Python Translator," An Internatonal Journal - CAIJ, vol. 5, pp. 1-18, 2018.
[3]. N. F. R. J. I. M. Christie Thottam, "Intelligent Python Code Analyzer [IPCA]," International Journal Of Creative Research Thoughts [IJCRT], vol. 12, no. 3, pp. 1-11, 2024.
[4]. N. M. S. S. J. L. R. Z. R. F. A. R. T. N. N. H. W. a. H. H. Chongzhou Fang, "Large Language Models for Cde Analysis: Do LLMs Really Do Their Job?," International Journals, vol. 1, pp. 1- 18, 2024.
[5]. X. X. X. Z. Y. L. P. D. Gang Fan, "Static Code Analysis in the AI Era: An In depth Exploration of the concept, Function, and Potential of Intellgent Code Analysis," Ant Groups, China, vol. 1, pp. 1-13, 2023.
[6]. E. J. a. Y. Sun, "An Algorithm-Adaptive Source Cde Converter to Automate The Translation From Pythn To Java," AIRCC, pp. 1-15, 2021.