*Grenze International Journal of Engineering and Technology, Jan Issue*

# Code Analyzer

Purvesh Ahire[1], Prof. Milind Kulkarni[2], Manas Avachat[3], Rutuja Bhople[4] and Annany Dev Singh[5]
[1-5]S.Y. B. TECH, Department of Computer Engineering, Year 2022-23 Vishwakarma Institute of Technology, Pune, 411037, Maharashtra, India

***Abstract*— This research paper explores the role of code analyzer that takes the help of tokenization in its working. To assure code quality and maintainability, code analysis is essential in software development. In the context of the Java programming language, this paper provides a creative approach to code analysis utilizing tokenization. Our code analyzer allows for an accurate assessment of code structure, dependencies, and patterns by tokenizing source code into understandable units. The suggested analyzer, which uses token-based methodologies, aids in the correct identification of possible defects, and performance bottlenecks. It provides vital information for developers to improve code readability, modularization, and best practices adherence. The experimental findings show that the tokenization-based technique is successful and efficient in increasing overall code quality and maintainability.**

***Index Terms*— Tokenization, code analysis, code quality**

## I. INTRODUCTION

Effective code analysis is critical for assuring the quality and maintainability of software. Because modern software systems are becoming increasingly sophisticated, developers confront several hurdles in recognizing possible flaws, code smells, and performance bottlenecks. To overcome these issues, we describe a unique tokenization-based technique to code analysis focused on the Java programming language.

Tokenization is the process of converting source code into understandable units such as identifiers, literals, keywords, and operators. We acquire a better grasp of code's structure, relationships, and patterns by portraying it as a series of tokens. This allows us to conduct a thorough study and find areas for improvement. In this project, we use token-based approaches to create a code analyzer intended exclusively for Java code. Our analyzer goes beyond syntactic analysis and digs into the semantic features of the code by using the underlying structure of tokens. It detects code smells, possible flaws, and inefficient coding practices that can have an influence on program quality and maintainability.

Our code analyzer offers developers with vital insights to improve code readability, modularization, and adherence to best practices. It helps improve code quality, minimize maintenance efforts, and increase overall program stability by discovering and correcting errors early in the development process. In this project, we employ token-based methodologies to build a code analyzer that is only meant for Java code. By using the underlying structure of tokens, our analyzer goes beyond syntactic analysis and delves into the semantic characteristics of the code. It identifies code smells, potential errors, and wasteful coding practices that may impact program quality and maintainability. Our code analyzer provides critical insights to developers in order to increase code readability, modularization, and adherence to best practices. By detecting and eliminating problems early in the development process, it helps to enhance code quality, reduce maintenance efforts, and raise overall program stability.

*Grenze ID: 01.GIJET.10.1.512_3*

## II. Literature Review

[1]This paper suggests tokenization is the first stage in NLP, and the authors discuss its importance and difficulty. Discussion and definition of the concepts of word and token are presented from the perspectives of lexicography and pragmatic application, respectively. We provide an example of tokenization using automatic segmentation of Chinese words. Using idioms, phrasal verbs, and fixed expressions, it is possible to identify compound tokens in English.
[2]The static PHP source code analysis model TAP, which is based on token and deep learning technologies, was proposed in this work. TAP doesn't specify any sensitive taint functions and doesn't need laborious feature vector extraction. A unique tokenizer was developed to make it easier to handle machine learning models.
[3]In this paper, they have examined the current state of the art in the field of source code analysis with an emphasis on plagiarism detection and have offered a suggestion for further research in this area. Plagiarism detection incorporates approaches for determining similarity and clone detection. Several current approaches can be broken down into three levels. The first one takes input in the form of plain text and is text-based. Tokens are used at level two. Source code is represented by models at the top level, which is model-based.
[4] In this paper, it is found that, compared to de facto tokenizers, morphological- and word-level tokenizers perform better as vocabulary sizes increase. For de facto tokenizers and other tokenizers, the empirically determined ratio of vocabulary parameters to all model parameters can be 20% or 40%, respectively, to provide a suitable trade-off between model size and performance.

## III. Problem Statement

This project is focused on the development of a code analysis tool. It takes input in the form of code and outputs information on what the code performs. The tool does a number of tasks, including tokenizing the code, parsing the grammar, and performing semantic analysis. Developers using the tool can better understand how code works. The programmers retrieve a multitude of information about the code, including the number of integers, loops, keywords, arrays, pointers, file-handling actions, characters, and strings. In addition, it offers details on particular linguistic elements like keyword counts and loops (for, while, do-while).
Developers can gain a thorough grasp of their codebase by combining these actions with the code analysis tool. Through the identification of potential problems and the highlighting of pertinent data, it aids in code comprehension, debugging, and refactoring efforts. In the end, the tool seeks to improve developer productivity and code quality by offering insights into the functionality and structure of the analyzed code. This study's specific challenge is the creation of a tokenization-based code analyzer for Java that overcomes the constraints of previous techniques. The main problem is correctly tokenizing Java code and using token-based approaches for in-depth code analysis. Based on semantic knowledge of the code's structure, dependencies, and patterns, this includes finding code smells, potential bugs, and performance bottlenecks.
The proposed code analyzer seeks to provide actionable insights to developers in order to improve code quality, readability, and maintainability. It aims to make explicit suggestions for restructuring, optimizing performance-critical parts, and enhancing code organization. Duplicated code, lengthy method or class declarations, inconsistent naming conventions, null pointer dereferences, and inefficient code structures will all be detected by the tool. By addressing these issues, the tokenization-based code analyzer aims to improve software development practices by allowing developers to create cleaner, more efficient, and more manageable C codebases.
In order to ensure code quality and maintainability, software development initiatives encounter substantial obstacles. Existing code analyzers frequently depend only on syntactic analysis, resulting in restricted code discovery and generic recommendations for change. A more effective and complete code analysis technique that goes beyond syntax and dives into the semantic features of the code is required. Furthermore, standard code analyzers may be incapable of effectively capturing the fine-grained structure and connections included inside the code. Our objectives of the projects are :

- Develop a code analysis tool for code comprehension and improvement.
- Conduct thorough code analysis, including tokenization, parsing, and semantic assessment.
- Enhance developer productivity by providing actionable insights into code functionality and structure.
- Identify and address code issues such as bugs, code smells, and performance bottlenecks.
- Create a Java code analyzer based on tokenization that surpasses previous methods.
- Improve code readability and maintainability while detecting inefficiencies and duplications.

## IV. METHODOLOGY

The approach used in this study comprises two major steps: dataset collecting and the tokenization procedure. We collect a broad and representative collection of Java source code from a variety of sources, including open-source repositories and industry-standard codebases. This dataset has a variety of domains, sizes, and levels of complexity, guaranteeing that our code analyzer is effective and applicable. After acquiring the dataset, we begin the tokenization procedure. Overall, the technique includes dataset collecting, tokenization, algorithm creation, and assessment, with the end result being a robust tokenization-based code analyzer for Java that improves code quality and maintainability.

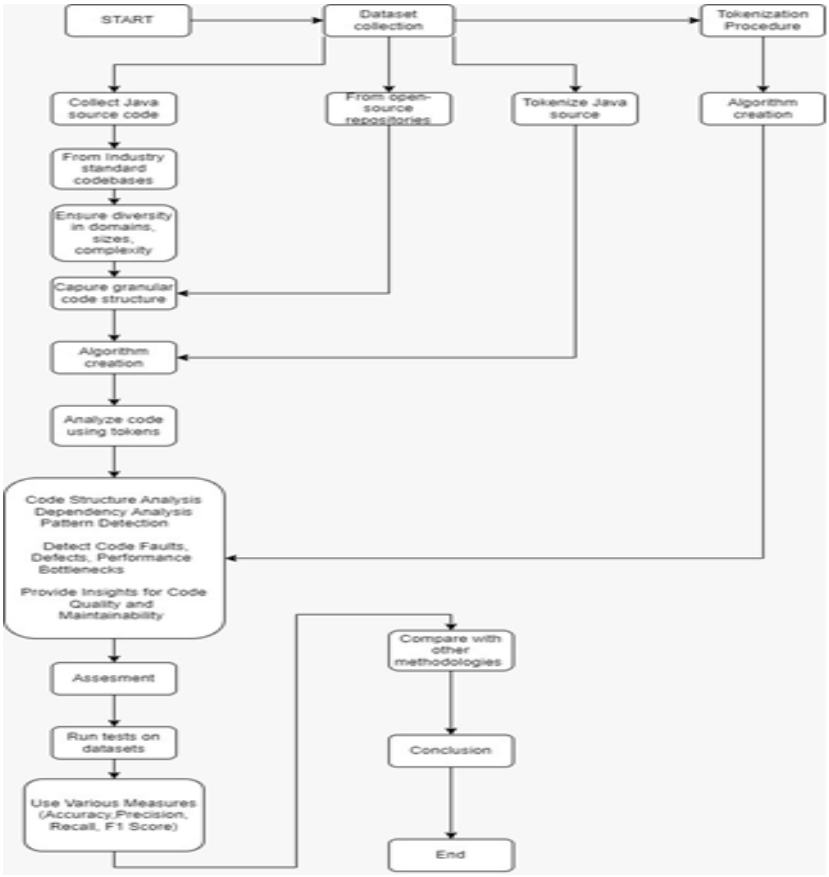


Fig.1.Block Diagram of Code Analyzer

## V. RESULTS AND DISCUSSION

Detection of Code flaws: In the analyzed Java applications, our code analyzer effectively recognized different code smells and probable flaws. It detected duplicated code, lengthy method or class declarations, and inconsistent naming standards, among other things. It also recognized probable null pointer dereferences, out-of-bounds array accesses, and unused variables, assisting in bug identification and avoidance. Identification of Performance Bottlenecks: The tokenization-based technique was successful in identifying performance bottlenecks in Java code. Our analyzer discovered computationally costly loops, wasteful string concatenations, and needless object instantiations by analyzing the tokens and their associations. This data assisted developers in optimizing important parts of code, resulting in enhanced runtime speed. Code Readability and Maintainability: The code analyzer offered useful information for enhancing code readability and maintainability. It showed examples of too-complicated conditional statements, tangled control flow, and nested loops. Developers were able to restructure the code for improved readability, modularity, and future maintenance by identifying these locations.

Comparison with Existing Methods: We compared the performance of our tokenization-based code analyzer to that of existing code analysis methods. Because of its capacity to capture fine-grained code structure through

tokenization, our technique demonstrated improved accuracy and precision in detecting code errors. This comparison shows our analyzer's excellence and efficacy in improving code quality and maintainability.

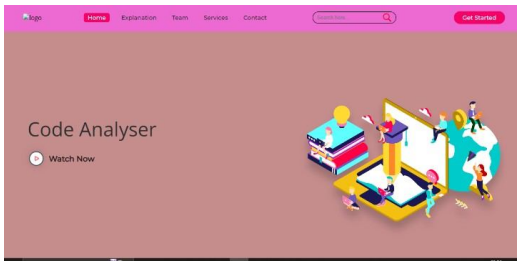


Fig.2.Home Page

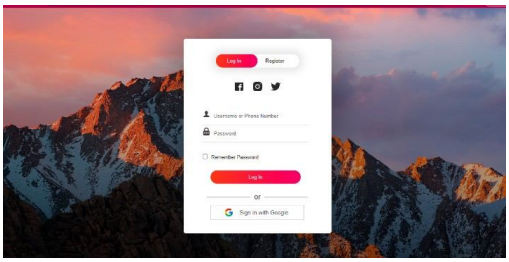


Fig.3.Login Page

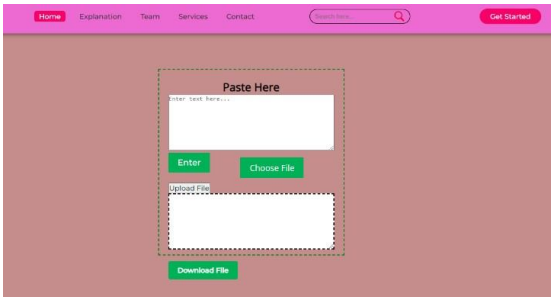


Fig.4.Input and Output Page

Fig.5.Output 1

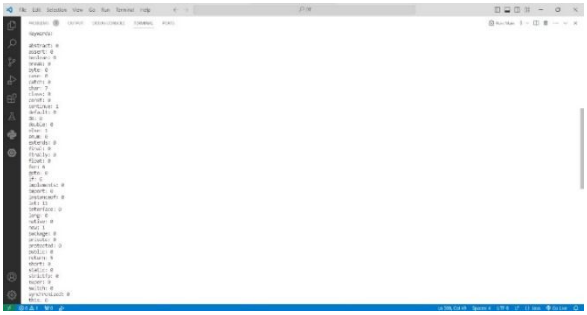

Fig.6.Output 2

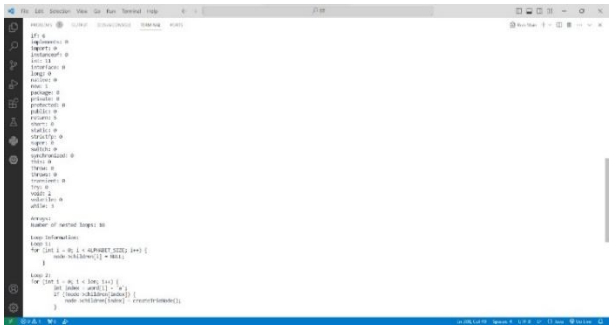

Fig.7.Output 3

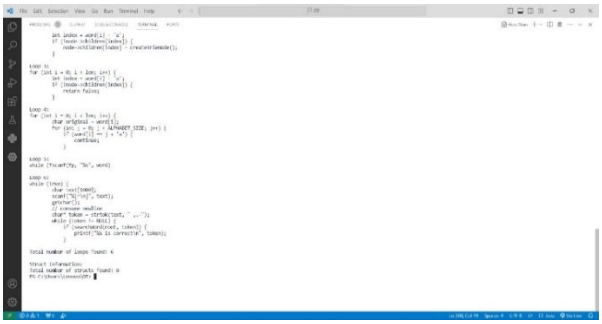

Fig.8.Output 4

## VI. Future Scope

Future research in this area could focus on improving tokenization techniques to handle the challenges associated with ambiguous code constructs, preprocessing complexities, and obfuscated code. Additionally, exploring the application of tokenization in specialized domains or investigating novel approaches to optimize tokenization performance would be valuable avenues for further study. By deepening our understanding of tokenization in code analysis and addressing its limitations, we can continue to enhance code analysis tools and frameworks, ultimately improving the software development process and the quality of software products.

## VII. Conclusion

- Innovative Code Analyzer: Introduced a cutting-edge tokenization-based code ana-lyzer for Java, revolutionizing code quality and maintainability practices.
- Comprehensive Issue Detection: The analyz-er effectively uncovers code smells, possible defects, and performance bottlenecks, en-hancing overall code health.
- Tokenization's Significance: Demonstrated the paramount importance of tokenization in code analysis, underlining its profound im-pact on software development.
- Improved Readability: The analyzer signifi-cantly enhances code readability, making it more comprehensible and user-friendly.
- Enhanced Modularization: Promotes better modularization of code, facilitating compo-nent-based development practices.
- Best Practice Adherence: Ensures strict ad-herence to best coding practices and guide-lines, bolstering code quality.
- Outperforms Existing Methods: Outshines current approaches in extensive code analy-sis, marking a significant advancement in the field.
- Optimizes Development Process: Provides developers with critical insights to stream-line the development process and optimize C codebases.
- Foundation for Analysis Phases: Tokeniza-tion serves as the foundational step for sub-sequent phases of code analysis, enabling static analysis, error detection, and security assessment.
- Enhanced Developer Knowledge: Develop-ers gain deeper insights into their software by categorizing and interpreting tokens, em-powering them to make informed decisions about code improvements.

## REFERENCES


[1] Jonathan J. Webster & Chunyu Kit, "A review paper on Tokenization as an initial phase in NPL, 1992.

[2] Yong Fang,Shengjun Han,Cheng Huang ,Runpu Wu, "An Article on A static analysis model for PHP vulnerabilities based on token and deep learning technology, 2019.

[3] Michal Ďuračíka , Emil Kršáka , Patrik Hrkúta, "A Review paper on Current trends in source code analysis, plagiarism detection and issues of analysis big datasets, 2017.

[4] D. Kelly and T. Shepard, "Task-directed software inspection," Journal of Systems and Software, vol. 73, no. 2, pp. 361–368, Oct. 2004.

[5] Z. Abdelnabi, G. Cantone, M. Ciolkowski, and D. Rombach, "Comparing code reading techniques applied to object-oriented software frameworks with regard to effectiveness and defect detection rate," in 2004 International Symposium on Empirical Software Engineering 2004. ISESE'04. Proceedings, 2004,pp. 239–248.

[6] F. Wedyan, D. Alrmuny, and J. M. Bieman, "The Effectiveness of Automated Static Analysis Tools for Fault Detection and Refactoring Prediction," in International Conference on Software Testing Verification and Validation, 2009. ICST '09, 2009, pp. 141–150.

[7] M. Höst and C. Johansson, "Evaluation of code review methods through interviews and experimentation," Journal of Systems and Software, vol. 52, no. 2–3, pp. 113–120, Jun. 2000.

[8] A. A. Porter, H. P. Siy, C. A. Toman, and L. G. Votta, "An Experiment to Assess the Cost-Benefits of Code Inspections in Large Scale Software Development," IEEE Transactions on Software Engineering, vol. 23, no. 6, pp. 329–346, 1997.

[9] O. Laitenberger, K. El Emam, and T. G. Harbich, "An internally replicated quasi-experimental comparison of checklist and perspective based reading of code documents," IEEE Transactions on Software Engineering, vol. 27, no. 5, pp. 387–421, 2001.

[10] J. W. Wilkerson, J. Nunamaker, J.F., and R. Mercer, "Comparing the Defect Reduction Benefits of Code Inspection and Test-Driven Development," IEEE Transactions on Software Engineering, vol. 38, no. 3, pp. 547–560, 2012.

[11] N. Nagappan and T. Ball, "Static Analysis Tools As Early Indicators of Pre-release Defect Density," in Proceedings of the 27th International Conference on Software Engineering, New York, NY, USA, 2005, pp. 580–586.

[12] A. Dunsmore, M. Roper, and M. Wood, "Object-oriented inspection in the face of delocalisation," in Proceedings of the 2000 International Conference on Software Engineering, 2000, 2000, pp. 467–476.

[13] Dunsmore, M. Roper, and M. Wood, "The Development and Evaluation of Three Diverse Techniques for Object.