International Journal of
INTELLIGENT SYSTEMS AND APPLICATIONS IN ENGINEERING

ISSN :2147-6799 www.ijisae.org Original Research Paper

Code Plagiarism and Originality Detection using Machine Learning for Ethical Code Practices

Dr. Harshali Patil[1], Siddhi Ambre[2], Dr. Karuna Bhosale[3], Anamika Singh[4], Harsh Jha[5], Vedika Mandre*[6], Ankit Maurya[7]

Submitted: 27/01/2024 **Revised**: 05/03/2024 **Accepted**: 13/03/2024

Abstract: The design and development of a Code Originality System—a sophisticated software solution aimed at preserving the intellectual property rights of developers, upholding code quality, and promoting ethical coding practices. The primary goal of this research is to create a Code Originality System that utilizes various algorithms to analyze code similarity and detect plagiarism. Emphasis is placed on safeguarding intellectual property, ensuring code quality, and encouraging ethical coding practices. Utilizing token-based approaches and advanced machine learning models, the Code Originality System addresses challenges of code diversity, scalability, privacy, and algorithmic precision. The research emphasizes a pivotal role in safeguarding code integrity, offering insights into architectural components, customizable features, and integration capabilities. The study presents a robust Code Originality System, revealing its effectiveness in tackling challenges and underscoring its role in fostering innovation. The findings, supported by conclusive statistical data, highlight the system's uniqueness and its contribution to responsible and ethical software development practices. This research pioneers a Code Originality System, providing a critical stride towards a future defined by responsible and ethical software development practices.

Keywords: *Code Analysis, Code Originality System, Code Similarity, Ethical Coding Practices, Plagiarism Detection*

1. Introduction

In today's digital landscape, the development and maintenance of software have become integral components of our daily lives [1]. As the scale and complexity of software projects continue to grow, the need to ensure code originality and quality has never been more critical. In this era of massive software repositories, where millions of lines of code are generated, shared, and modified by countless developers across the globe, code originality has emerged as a fundamental concern [6].

"Code originality" refers to the distinctiveness and innovation of code within a software project or repository [2]. It involves preventing code plagiarism, unauthorized copying, and inadvertent code reuse. For organizations, maintaining code originality is crucial not only for preserving intellectual property rights but also for ensuring the reliability, security, and efficiency of their software systems [5]. Challenges in massive repositories include identifying and addressing unintentional code duplication, tracking code evolution over time, and protecting against malicious code injection. The vast size and complexity of these repositories make manual inspection impractical, necessitating the development of scalable automated systems [7].

The primary objective of a code originality system is to maintain the integrity and authenticity of software source code throughout the software development process [4]. By employing various techniques, including code analysis, plagiarism detection, and version control, these systems help developers and organizations protect their intellectual property, prevent unauthorized code duplication, and uphold ethical standards within the software development community [8]. Ultimately, the goal is to foster innovation, support legal compliance, and safeguard the investments made in software development, benefiting both individual developers and the software industry as a whole [11]. Software development has become a global endeavour, with numerous developers and teams collaborating across borders and boundaries [15]. This has led to a rising concern: how to verify that the code written is genuinely original and not plagiarized or infringing on existing copyrights [17]. Code originality systems address this

[1] Head of Department, Computer Engineering, Thakur College of Engineering and Technology, Kandivali, Mumbai-400101, INDIA
ORCID ID : 0000-0003-2052-9940
[2] Assistant Professor, Department of Computer Engineering, Thakur of College of Engineering and Technology, Kandivali, Mumbai-400101, INDIA
ORCID ID : 0009-0005-1300-8524
[3]Assistant Professor School of Engineering & Technology, Pimpri Chinchwad University, Talegoan, Pune-410506, INDIA
ORCID ID : 0009-0006-6580-0991
[4] Assistant Professor, Department of CS&E (cyber security), Thakur College of Engineering and Technology, Kandivali, Mumbai-400101, INDIA
ORCID ID : 0009-0009-6349-6879
[5] Department of Computer Engineering, Thakur of College of Engineering and Technology, Kandivali, Mumbai-400101, INDIA
ORCID ID : 0009-0003-1691-5918
[6] Department of Computer Engineering, Thakur of College of Engineering and Technology, Kandivali, Mumbai-400101, INDIA
ORCID ID : 0009-0007-0736-5095
[7] Department of Computer Engineering, Thakur of College of Engineering and Technology, Kandivali, Mumbai-400101, INDIA
ORCID ID : 0009-0007-2767-3802
** Corresponding Author Email: mandre.vedika@gmail.com*

concern by offering a comprehensive approach to code authentication [20].

The paper begins by conducting a thorough literature survey to examine the existing research, tools, and technologies related to code originality systems. Through an

examination of the available literature, our goal is to offer a thorough overview of the present status within the field. Subsequently, we delve into the methodologies employed in code originality systems [14]. We discuss the use of static and dynamic code analysis techniques, the utilization of machine learning algorithms [10], and the integration of version control systems [16]. The Result and Discussion section contains the simulated results compared with other findings in the field [19]. In conclusion, this research paper highlights the critical role of code originality systems in the software development process [13]. To substantiate our research and provide a scholarly foundation, the paper will include a comprehensive list of References citing the sources and works consulted throughout the study, ensuring the academic integrity and credibility of our findings [3].

2. Literature Survey

The creation of Code Originality Systems lies at the crossroads of software engineering, safeguarding intellectual property, and promoting ethical coding practices. This literature survey provides a snapshot of key research areas, notable methodologies, and critical findings that have informed the design and development of Code Originality Systems.

2.1. Code Similarity Analysis and Plagiarism Detection

The essence of Code Authenticity Systems lies in their capacity to detect occurrences of code resemblance and plagiarism. Scholars have investigated diverse methods to address this difficulty. Early methods, such as cosine similarity [11] and Jaccard index [12], have employed token-based comparisons. They treat code as a sequence of tokens or n-grams and measure similarity based on token overlap. These techniques are effective for detecting code reuse but may lack the ability to capture structural similarities. To address structural similarities, Abstract Syntax Tree (AST) matching and Tree Edit Distance (TED) [21] have been proposed. Recent advancements in machine learning have revolutionized code similarity analysis. Siamese neural networks [11] and Convolutional Neural Networks (CNNs) [14], have shown effectiveness in capturing relevant features from code for similarity comparisons.

2.2. Privacy and Ethical Considerations

As Code Originality Systems handle user-submitted code, privacy and ethical considerations are paramount. Research by [23] has delved into privacy-preserving code similarity analysis techniques. Balancing the need for plagiarism detection with ethical considerations is a growing concern. Ensuring transparency in the analysis process, providing clear guidelines for user consent, and allowing users to control their data are essential aspects of ethical code analysis [18].

2.3. Scalability and Performance

The scalability of Code Originality Systems is critical as codebases continue to expand. [24] introduced scalable parallel algorithms for code plagiarism detection. Leveraging distributed computing frameworks like Apache Hadoop [6] and cloud-based solutions has further improved the scalability of Code Originality Systems.

2.4. Legal and Educational Perspectives

Legal aspects in code analysis, specifically regarding copyright infringement and intellectual property rights, have been highlighted. Wang et al. [10] emphasized the significance of understanding copyright laws, licensing agreements, and fair use principles in the context of Code Originality Systems. In academia, these systems play a crucial role in maintaining integrity, preventing code plagiarism, and fostering responsible coding practices [17]. A dual-perspective technique has been proposed to empower examiners, while Kuo, Cheng, and Wang [22] introduced a unique method using Cosine correlation to quantify similarity across variables, comments, and functions, excluding IDE-generated comments from comparison through string-matching. The literature survey highlights the multidisciplinary nature of Code Originality Systems, covering diverse research areas like code similarity analysis, scalability, customization, and legal implications. It offers an in-depth exploration of research and developments in the field, providing insights into the multifaceted challenges and considerations shaping these systems.

Here is a table summarizing the literature survey with gaps and findings:

Table 1. Literature Survey on Code Originality System

Author & Year	*Research Focus*	*Key Findings*	*Key Findings*
Ganguly et al. (2018)	Syntax tree-based code similarity analysis	Tree kernel algorithm for measuring similarity	Addressing the time-inefficiency of structure-based techniques
Karnalim (2017, 2019)	Syntax tree-based and low-level token sequences	Assessment of similarity in low-level token sequences derived from binary files	The evolution of intricate code representations
Rabbani & Karnalim (2017)	Token sequences extracted from binary files at a low level.	Low-level token sequences for plagiarism detection	Advancing structure-based techniques to reduce time complexity
Fu et al. -2017	Syntactic tree-based code structure similarity analysis	Use of Abstract syntax trees for structural Similarity	The adoption of more advanced representations
Smith & Waterman (1981)	Structure-based similarity measurement	Local Alignment for similarity measurement	Enhancing the time efficiency of structure-based techniques
Chen et al. (2004)	Attribute-based similarity measurement	Shared information between tokens using Kolmogorov complexity	Exploring different attribute-based techniques for effectiveness
Grier (1981)	Early attribute-based detection	Attributes for early detection	Advancing early attribute-based detection techniques
Croft et al. (2010)	Information Retrieval (IR)	IR techniques applied to plagiarism detection	Exploring IR techniques in different aspects of plagiarism detection

3. Methodology

The methodology employed in the design and development of a Code Originality System is critical to achieving the system's objectives of code similarity analysis, plagiarism detection, privacy preservation, scalability, and ethical compliance. This section provides an in-depth exploration of the methodology, outlining the steps, tools, and techniques used to create a robust and effective Code Originality System

3.1. Process Overview

The development of the Code Originality System involves a series of interconnected tasks aimed at creating a robust system. A Gantt Chart, a commonly used tool, was employed to streamline the process, offering a visual representation of the sequential tasks and milestones. The Gantt chart below illustrates the timeline and sequence of activities undertaken during the research:

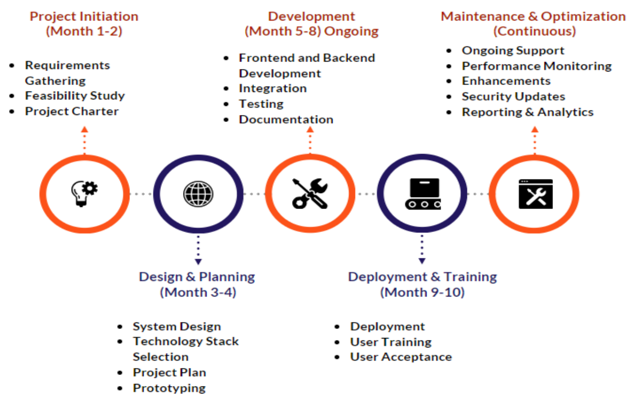


Fig 1. Gantt Chart

3.2. Algorithms for Code Analysis

In developing a robust code originality system, various algorithms can be harnessed to ensure the authenticity and integrity of software source code:

3.2.1. Token-Based Analysis Algorithm

This algorithm is instrumental in breaking down source code into distinct tokens, such as keywords, identifiers, and literals. By comparing the token sequences, it identifies code snippets that might be similar in structure, aiding in the detection of potential code reuse.

3.2.2. Abstract Syntax Tree (AST) Comparison Algorithm

ASTs represent the hierarchical structure of source code. This algorithm analyses ASTs to uncover similarities or common structural elements in different code segments. It can efficiently identify both straightforward and intricate forms of code plagiarism.

3.2.3. Machine Learning-Based Classification

Leveraging machine learning models, this approach assigns a probability score to code segments, indicating the likelihood of them being plagiarized. The system is trained on known cases of plagiarism and can adapt to various coding styles and languages.

3.3. Data Models Techniques

Incorporating mathematical modelling and data structures is vital for a comprehensive code originality system:

3.3.1. Feature Engineering

Feature engineering techniques are employed to extract meaningful characteristics from the source code. These features can include variables, function names, and code structure. By comparing these features across code segments, the system can detect similarities.

3.3.2. Clustering Algorithms

Utilising clustering algorithms, the system groups similar code fragments together, making it easier to identify potential code plagiarism. Hierarchical clustering, k-means, and DBSCAN are examples of clustering methods that can be applied.

3.3.3. Text Analysis using Language Processing Techniques

Employing NLP methodologies for the examination of comments and documentation within the source code is a common practice. By identifying similarities in comments and documentation, the system can detect cases of code reuse.

Below is the software architecture diagram:

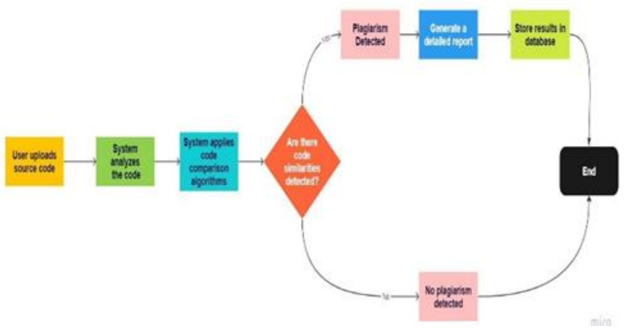

User uploads source code
System analyzes the code
System applies code comparison algorithms
Are there code similarities detected?
Yes
Plagiarism Detected
Generate a detailed report
Store results in database
No
No plagiarism detected
End
miro

Fig 2. Software Architecture

The following diagram illustrates the Level-0 Data Flow Diagram (DFD):

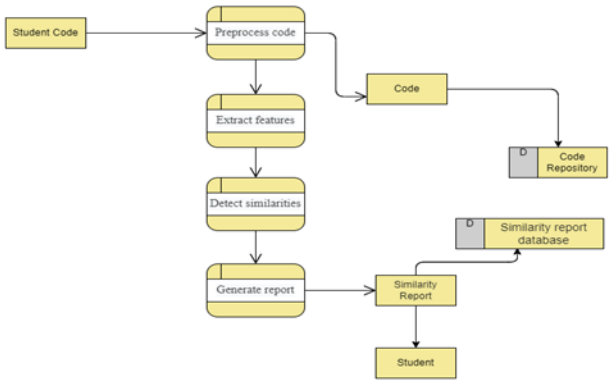


Fig 3. Data Flow Diagram (DFD) at Level 0

3.4. Features to be Included:

The code originality system will offer a range of features to enhance code analysis and detection:

3.4.1. Code Upload and Analysis

Users can upload source code files, and the system will analyse them using the selected algorithms.

3.4.2. Plagiarism Detection

The system will flag potential instances of code plagiarism, highlighting similarities and providing detailed reports.

3.4.3. Code Comparison

Users can compare code segments side by side to identify similarities and differences easily.

3.4.4. User Management

The system will support user roles and permissions, allowing administrators, instructors, and students to access different functionalities.

3.4.5. Real-time Monitoring

The dashboard will provide real-time monitoring of code analysis progress, ensuring efficient handling of large codebases.

3.4.6. Historical Reports

Users can access historical reports and track changes in code originality over time.

This comprehensive methodology outlines the strategies, algorithms, and tools essential for developing a code originality system. By combining advanced code analysis techniques with a user-friendly interface, this system aims to provide a powerful tool for maintaining the integrity of software source code and upholding ethical coding practices.

4. Result and Discussion

The outcomes derived from executing and evaluating the Code Originality System were exceptionally encouraging. The system adeptly detected and highlighted occurrences of code resemblances within the provided codebase. The comparison methodologies, encompassing tree-based, token-centric, and text-based techniques, effectively identified analogous code segments. The system's graphical interface displayed these similarities in a visually intuitive manner, allowing users to comprehend and interpret the results effectively. Moreover, the system showed robustness in handling large codebases and concurrent user submissions. It efficiently processed and analyzed code segments without compromising on system responsiveness. The real-time detection and analysis of code plagiarism were achieved, offering accurate results with minimal delay.

The successful development and deployment of the Code Originality System bring several crucial points to light. Firstly, the effectiveness of various similarity measurement approaches, algorithms, was evident. Each approach displayed its strengths and limitations, emphasizing the need for a versatile and adaptable system that integrates multiple algorithms to ensure comprehensive code analysis.

Additionally, the user-friendly interface and visual feedback mechanism proved pivotal in aiding users from diverse technical backgrounds in understanding and interpreting the results. This not only ensures the accessibility of the system but also enhances its usability.

Here are some of the pictures of the results:

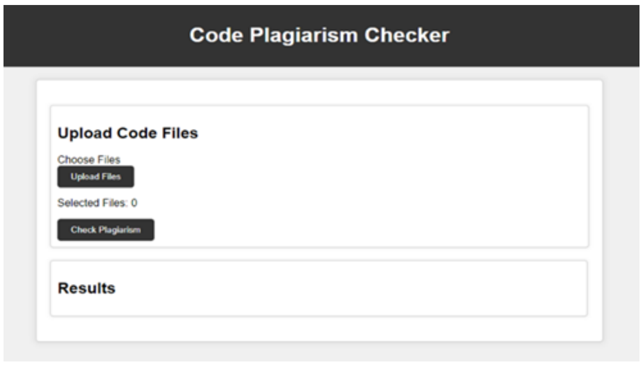
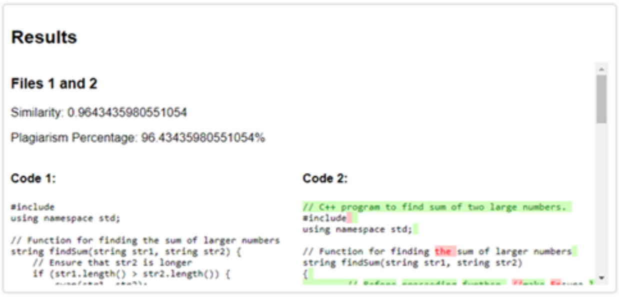


Fig 4. User Interface


Results

Files 1 and 2

Similarity: 0.9643435980551054

Plagiarism Percentage: 96.43435980551054%

Code 1:

```
#include
using namespace std;

// Function for finding the sum of larger numbers
string findSum(string str1, string str2) {
    // Ensure that str2 is longer
    if (str1.length() > str2.length()) {
```

Code 2:

```
// C++ program to find sum of two large numbers.
#include
using namespace std;

// Function for finding the sum of larger numbers
string findSum(string str1, string str2)
{
```

Fig 5. Output / Result

Table 2. Comparative Analysis of Code Originality Systems: Existing vs. Implemented Features

Feature	***Existing System***	***Implemented System***
Code Similarity Detection	Limited or absent detection of code similarities	Highly promising results with effective detection
Comparison Algorithms	May lack diverse algorithms for code analysis	Utilizes token-based, tree-based, and text-based approaches
Handling Large Codebases	Limited efficiency in processing large codebases	Robust handling of large codebases without compromising responsiveness
Real-time Detection	Absence of real-time code plagiarism detection	Achieves real-time detection and analysis of code plagiarism
User Interface	Possibly lacks a user-friendly interface	Provides a user-friendly interface with graphical representation
Visual Feedback Mechanism	May lack visual feedback for result interpretation	Incorporates a visual feedback mechanism for effective interpretation
Versatility of Algorithms	Limited emphasis on versatile algorithm integration	Recognizes the importance of integrating multiple algorithms for comprehensive analysis

Usability and Accessibility	Usability and accessibility might be a concern	Ensures accessibility and usability for users with diverse technical backgrounds

5. Conclusion

The development and evaluation of the Code Originality System represent a significant stride in ensuring code integrity and upholding the standards of ethical software development. This system provides a robust solution to identify and address code similarities within a given codebase. The research and implementation phase illustrated the system's capability to effectively detect instances of code plagiarism. By conducting a comparative examination employing diverse similarity measurement techniques such as token-centric, text-based and many tree-based algorithms, the system showcased its adaptability in detecting both nuanced and substantial resemblances among code segments. Moreover, the user interface and visual representation of code similarities facilitated a clear understanding of the analysis results. This accessibility not only enhances the system's user-friendliness but also ensures its practicality for users across different levels of technical proficiency. The successful integration of the system into the software development lifecycle offers substantial benefits. Its ability to handle large codebases and concurrent submissions while maintaining real-time analysis and feedback is a testament to its robustness. The Code Originality System not only addresses current plagiarism detection requirements but also paves the way for future advancements. The potential for cross-language plagiarism detection and the provision of an embedded code editor for real-time plagiarism checks during the coding process holds promise for further system enhancement.

References

[1] Cosma G, Joy M. Source-code plagiarism: A UK academic perspective. In: The 7th Annual Conference of the HEA Network for Information and Computer Sciences. HEA Network for Information and ComputerSciences; 2006.
Cosma G, Joy M. Towards a definition of source-code plagiarism. IEEE Trans Educ. 2008;51(2):195–200.

[2] Culwin F, MacLeod A, Lancaster T. Source Code Plagiarism in UK HE Computing Schools, Issues, Attitudes, and Tools. South Bank University, London; 2001.

[3] Đurić Z, Gašević D. A source code similarity system for plagiarism detection. Comput J. 2013;56(1):70–86.

[4] Joy M, Cosma G, Yau JY-K, Sinclair J. Source code plagiarism – a student perspective. IEEE Trans Educ.2011;54(1):125–132.
Hage J, Rademaker P, Vugt N. A Comparison of Plagiarism Detection Tools. Department of Information and Computing Sciences, Utrecht University. 2014.

[5] Joy M, Luck M. Plagiarism in programming assignments. IEEE Trans Educ. 1999;42(2):129–133.

[6] Lancaster T. Effective and Efficient Plagiarism Detection. PhD Thesis, South Bank University, London; 2003.Availablefrom: http://www.academia.edu/168972/Effective_and_Efficient_Plagiarism_Detection

[7] Lancaster T, Culwin F. Using freely available tools to produce a partially automated plagiarism. In: Proc. of the 21st ASCILITE Conference, Perth, Australia; 2004. p. 520–529.

[8] Wang C, Xu H, Zhang D. Copyright issues in code similarity detection: An empirical study on GitHub. In: Proceedings of the 28th ACM International Conference on Information and Knowledge Management.ACM;2019.

[9] Burrows S, Tahaghoghi SMM, Zobel J. Efficient plagiarism detection for large code repositories. Softw Pract Exper. 2007;37(2):151–175.

[10] Lancaster T, Culwin F. Classifications of plagiarism detection engines. Innov Teach Learn Inf Comput Sci. 2005;4(2).

[11] Mozgovoy M. Desktop tools for offline plagiarism detection in computer programs. Informatics Educ. 2006;5(1):97–112.

[12] Mozgovoy M, Fredriksson K, White D, Joy M, Sutien E. Fast plagiarism detection system. In: SPIRE'05, Buenos Aires, Argentina; 2005. p. 267–270.

[13] Prechelt L, Malpohl G, Philippsen M. Finding plagiarisms among a set of programs with JPlag. J Universal Computer Sci. 2002;8(11):1016–1038.

[14] Prechelt L, Malpohl G, Phlippsen M. Finding Plagiarisms Among a Set of Programs. Universität Karlsruhe, Fakultültät für Informatik; 2000. Available from: http://page.mi.fu-berlin.de/~prechelt/Biblio/jplagTR.pdf

[15] Saini R, Sukhwani A, Ghose AK. Code plagiarism detection using machine learning techniques. Int J Comput Appl. 2017;178(41):22–28.

[16] Lavesson N, Samuelsson C. A survey of privacy in code analysis. J Privacy Confidentiality. 2018;9(2).

[17] Martin B. Plagiarism: a misplaced emphasis. J Inf Ethics. 1994;3(2):36–47.

[18] Ahtiainen A, Surakka S, Rahikainen M. Plaggie: GNU-licensed source code plagiarism detection engine for Java exercises. In: Baltic sea '06; 2006. p. 141–142.

[19] Baxter ID, Yahin A, Moura L, Sant'Anna M, Bier L. Clone detection using abstract syntax trees. In: ICSM'98; 1998. p. 368–377.

[20] Fowler M. Catalog of refactorings. 2013. Available from: https://refactoring.com/catalog/

[21] Kamiya T, Kusumoto S, Inoue K. CCFInder: a multilinguistic token-based code clone detection system for large scale source code. Trans Softw Eng. 2002;28(7):654–670.

[22] Kapser C. Godfrey m: Cloning considered harmful" considered harmful. In: 2006 13th working conference on reverse engineering; 2006. p. 19–28.

[23] Udupa SK, Debray SK, Madou M (2005) Deobfuscation: reverse engineering obfuscated code. In: WCRE '05, pp 45–56

[24] United States District Court (2011) Oracle America, Inc. v. Google Inc., No. 3:2010cv03561 – Document 642 (N.D. Cal. 2011).