IJCRT.ORG ISSN : 2320-2882

INTERNATIONAL JOURNAL OF CREATIVE RESEARCH THOUGHTS (IJCRT)

An International Open Access, Peer-reviewed, Refereed Journal

BUILDING A WEB-BASED PLAGIARISM SYSTEM USING ABSTRACT SYNTAX TREE AND GREEDY STRING TILING ALGORITHM

Hajara John Garba[1], Gilbert Aimufua[2], [3]Name of 3rd Author
[1]IT Centre Nasarawa state university, Keffi Nigeria
[2]Department of Computer Science, Keffi Nigeria

Abstract: Plagiarism detection is the process of looking for similarities in documents that are electronic-based. In the academic community, academic integrity is a delicate subject. Consequently, it is vital to fiercely fight them. But plagiarism is a problem everywhere. Because there are so many documents on the internet, and it is possible to copy and paste. The early stages of plagiarism detection involved either manual detection or comparison to previously examined sources. The most crucial aspect of this study is the development of a reliable system that can test a single source of code against a sizable external code base and assess how similar the results are. It should be a system that can compare offline and online source code, as most system comparisons are done from offline to offline. However, this new system will compare both. To protect it against fragility and to enable quick comparison of huge source code, an abstract syntax tree will be modified. This would significantly lessen the problem of students plagiarizing in their assignments and research projects. The objective of this paper is to develop a method to identify instances of plagiarism in sets of source codes or texts that are submitted as part of student assignments or comparable scenarios. The proposed approach does not rely on external repositories, thereby enabling local assessment of the originality of the submitted work. To achieve this goal, the study focuses on various programming languages, including Java, Kotlin, C++, Python, and CSharp. By analyzing the syntax and content of the codes or texts, the method can accurately detect possible instances of plagiarism.

Keywords: Plagiarism, Abstract Syntax Tree, Parse, Tokens Tracer.

I. INTRODUCTION

With the use of internet search engines, it is simple to access a wide variety of websites that offer helpful information for writing academic essays. In many cases, these websites have been created by other academic institutions for the benefit of their own students. (Austin 2019). Following the recent global Covid-19 outbreak, the educational profession has further branched out into the internet space. Students finish a lot of assignments at home and send them to their teachers to be graded. The independent preparation of the assignments by the student, the lack of plagiarism, and the unlawful use of previous work must be objectively verified. It takes a lot of effort to individually check each one for originality, and catching cheating pupils isn't always doable. It takes a lot of time to independently prepare each assignment and to manually examine each one for originality. Additionally, it is not always possible to spot pupils who are cheating. Plagiarism is an extremely serious issue in academic contexts. The fact that you can copy and paste text from a variety of online resources so quickly makes it worse. Because the perpetrator stole and misrepresented someone else's work as their own, resulting into academic fraud. It speaks about a person's integrity and honesty. This is a common occurrence, especially in programming courses where it is simple to clone a successful solution. Some students simply duplicate someone else's work without crediting the original author because they believe that working on the assignment may not benefit them. Students also frequently work in groups, so they do not perceive a breach of this kind provided that everyone has the same solution (Ullah et al, 2019).

However, different people frequently have varied interpretations of what plagiarism is. For instance, it is normal practice to use the source code of a program created especially for a firm without authorization. (Zhang et al, 2019). In a written document, the standard text, tables, flowcharts, picture captions, and code can all contain plagiarism. Plagiarism can be committed by simply copying, paraphrasing, or obscuring the language without giving the author proper credit. By using clever editing techniques like synonyms, rendering, restructuring, summaries, translations, etc., the material can be changed. The degree of plagiarism in a work can range from simple copying and pasting to heavily translate and disguised language. technologies are being used by Detection Model to ensure that none of the documents have been copied, protecting the publishers' copyrights in the process. (Wang et al, 2018). There are various types of plagiarism, and in order to detect them, distinct policies must be established by specific professors, schools, faculties, and journal publishing companies. Plagiarism for the benefit of others is a serious infraction that may result in you losing points for the work you plagiarized, being placed on academic probation, or even being suspended or kicked out of your program or institution. Understanding what plagiarism is and how to properly attribute each author whose work you utilize in your own writing are the best ways to avoid being accused of (or accidentally committing) it (Kramer, 2022).

.1 Plagiarism Type

There are two types of plagiarism which are source code and textual plagiarism.

1.2 Source Code Plagiarism

University students often commit this form of plagiarism, which is difficult to identify. Students attempt to copy in full or in part the source code written by someone else as their own. The reuse of someone source code without providing adequate acknowledgment is known as source code plagiarism.

1.3 Textual plagiarism

This type of plagiarism often involves the creation of materials that are identical to or comparable to the original documents, reports, essays, scientific papers, and artwork by students or researchers at academic institutions

II. LITERATURE REVIEW

To cope with varying levels of plagiarism, from copy-paste to high-level plagiarism, Shrestha and Solorio (2013) presented a solution using n-grams with different properties**.**

The longest common sequence technique was designed to find semantic similarity in code. This fuzzy matching technique is integrated with this method to extract the longest common sequence structure in a chunk of codes (Ullah etal, 2018).

According to (Fu et al. 2017), the Abstract Synthetic Tree (AST) is used to capture abstract perspectives of various source codes. Using a High-level Fuzzy Petri net (HLFPN) based on AST, these traits are used to predict source code plagiarism and propose a novel technique to detect reused codes in students' programming projects.

In the work of Jhi et. al. (2015), the authors described three steps recognition algorithm built on abstract parse tree method to verify clones between C programs. The proposed source code clone extraction algorithm has three phases' elementary, classification and generalization. Their search is applied on a datasets reserved from scholars' programming. assignments. The distinguishing of code clones in a programming language is an important portion of software maintenance.

Over time, there has been an increased requirement for any news piece to be used in a functional program by the researcher and the institution that will publish it.

Gupta et al. (2011) concentrated on paraphrase utilized in PD from both cross-lingual and monolingual points of view. Through a deeper examination of the performance of the (Vector Space Model), the difficulties of the detection process were explored.

In (Ahuja, et al, 2020) Created a system that employed an extrinsic PD technique that was inspired by cognition, using semantic information to identify copied content without the requirement for human interaction.

Application of stylometry to computer code to assign authorship to anonymous binary or source code is known as programming authorship. It frequently entails dissecting and analyzing the particular patterns and traits of the programming code, then contrasting them with known-author computer code (Claburn, Thomas 2018)

Expert judgment on the levels of similarity and difference between code fragments may be provided based on the general appearance of the code or the use of programming idioms (MacDonell et al. 2019).

III METHODLOGY

The research methodology adopted in this paper focuses on a web-based source code and text plagiarism detection system in computer programming. Methodology incorporates the use of an Abstract Syntax Tree (AST) and the Greedy String Tiling Algorithm for plagiarism detection. The process involved in the architecture of the system is outlined as follows:

i. **Source Codes:** The system takes source code inputs, which are the pieces of code that need to be analyzed for potential plagiarism.

ii. **Parse and AST Generator using (ANTLR4):** The source code is parsed and converted into an abstract syntax tree using the ANTLR4 tool. ANTLR4 is a powerful parser generator that can handle various programming languages and create ASTs based on the code's grammar.

iii. **AST Modification:** After generating the AST, the system may perform modifications or pre-processing steps on the tree to standardize the code representation or apply certain optimizations.

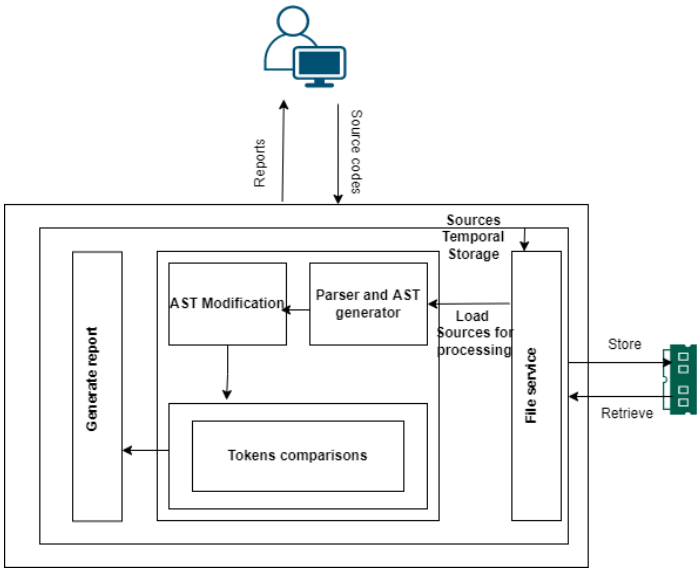
iv. **Tokens Comparisons:** The ASTs are then converted into sequences of tokens, representing various code elements like identifiers, literals, and operators. The system compares these token sequences to identify similarities and potential instances of plagiarism.

3.1 Modification of Abstract Syntax Tree (AST)

From Figure1

The system architecture shows some of the components that lead to the processes, starting from the first phase where sources were parsed and AST is been generated from the source's codes using ANTLR4 (In computer-based language recognition, ANTLR4 (pronounced antler). Another Tool for Language Recognition, is a parser generator that uses LL (*) for parsing. ANTLR4 is the successor to the Purdue Compiler Construction Tool Set (PCCTS), first developed in 1989, and is under active development) and the AST is further passed to the next phased for modification. In the modification phase, some irrelevant nodes such as nodes that are common to all the sources (starting nodes) from the AST are removed or replaced. Some parts of the nodes such as TOKEN values that are literals are removed and replaced with empty string.

The next phase is the similarity check which uses the Greedy String Tiling Algorithm introduced by Wise. And finally, the last step is report computation. In this phase the system group all the works that seem similar base on the grouping threshold the report will be provided.



3.2 Algorithm for Comparison

1. Load source codes from source
2. If sources are not files
   Process the sources without temporary storage
3. Otherwise
   Store the source in a temporary location
   Load the sources by combination with the size of 2
   Process the sources
   4. Return report

3.3 Detailed Flow of the Comparison

When comparing two strings A and B, the aim is to find a set of
Figure 1 System Architecture Diagram

sub-strings that are the same and satisfy the following rules: Only one token from B must match every token from A. This criterion means that portions of the source material that have been copied in a plagiarized program cannot be perfectly matched. Substrings can be detected wherever they exist in the string. According to this criterion, an assault that involves rearranging portions of the source code is ineffective.

Short substring matches are less dependable than long ones, hence long substring matches are desired. Short matches have a higher chance of being fictitious. When the third rule is successively applied for each matching step, a greedy algorithm with two phases results:

Phase 1:

The two strings are compared in this phase to find the most extensive contiguous matches. Three nested loops are used for this: In the first, all of the tokens in string A are iterated through; in the second, this token T is compared to each token in string B. The innermost loop tries to make the match as long as it can be if they are identical. The list of the longest common substrings is gathered by these nested loops.

Phase 2: Phase 2 assigns a mark to each maximal length match discovered in Phase 1. This indicates that all of their tokens have been tagged and cannot be utilized for more Phase 1 matches in a later iteration. This satisfies the first criteria from above by designating all the tokens a match becomes a tile and ensures that each token will only be utilized in one match.

Some of the matches might also overlap. In this instance, the first match discovered is picked, with the others being disregarded. Up until no more matches are discovered, these two phases are repeated. The method is guaranteed to end since each step sees a decrease in the length of the maximal matches of at least 1. Matches of just a few tokens would frequently happen by chance if matches of any length were permitted. Therefore, a minimum match length, also known as the "Minimum Match Length," is defined to prevent erroneous matches.

V. RESULT AND DISCUSSION

Figure 1 shows the fundamental interface of the application which provides users with a variety of options to choose from. The user can select the type of action they want to perform, whether it is text or source code comparison. Additionally, they have the option to choose between single or bulk mode depending on their needs and preferences. Finally, language selection is also available to ensure that the results are accurate and relevant to the user's requirements. These various choices give users greater flexibility and control over how they use the application in order to achieve their desired outcomes. The outline below captures the supported operations

A. Single code comparison (Requires two source codes)

B. Bulk comparison (Any number of source codes)
C. Selection of comparison mode (Source code or Text)
D. Threshold (disabled by default)
E. Languages selection and
F. Button to trigger the comparison operation

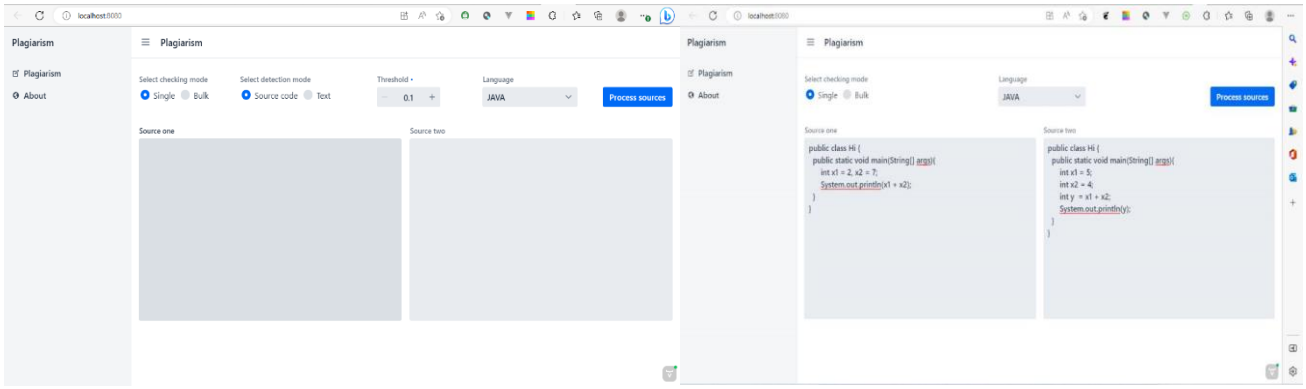


Figure 2: Fundamental Interface of the Application

Figure 3 Showing Single Mode Interface

The system has the capability to provide support for two different modes of detection, namely Single and Bulk. When choosing the Single mode, users are able to investigate plagiarism on just two source codes. The interface displayed depicts the gathering of source codes in this particular mode. There are two text areas that permit users to input sources based on their language mode preference. Once both sources have been provided and a language has been selected, users can initiate processing by clicking "process." During this process, the system will identify any instances of plagiarism present within the source codes. It is worth noting that this mode operates in a straightforward manner as both sources are parsed and processed directly without being temporarily stored in a directory.

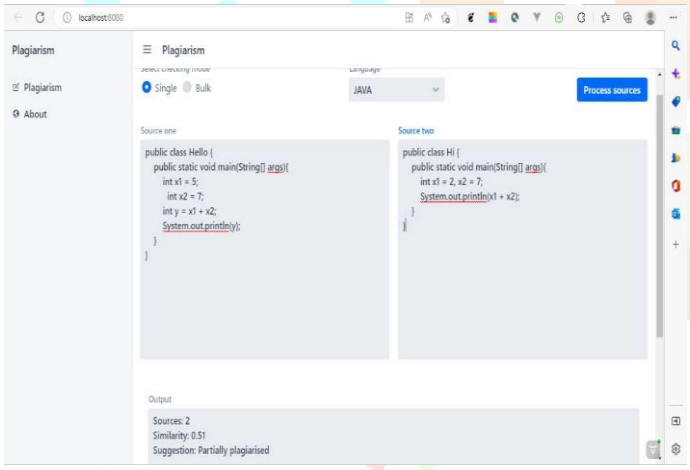


Figure 4: Interface showing Sample of Output

The single source/text comparison mode allows the user to check for plagiarism between two texts or sources and the Engine used depends on the Language selected. This takes the two sources or texts and as shown in the architecture diagram, the system saves these two files on a temporary file storage system before being loaded by the Parser for parsing and comparison. The ensuing report comprises of the following:

Sources: The total number of source codes processed for checking plagiarism

The similarity: The score of the similarity check between the sources codes and the score is computed on a scale of 0-1.

Suggestion: This suggests if there is plagiarism or not based on the similarity score computed, it could be not plagiarized, partially plagiarism, plagiarized and totally plagiarized

The information presented below offers a thorough and comprehensive analysis of the results obtained from the tests conducted on various programming languages that were used in this study. This data has been meticulously collected and compiled to provide valuable insights into the performance and effectiveness of different programming languages. The analysis takes into account several key factors, including but not limited to, speed, efficiency, reliability, and scalability. Through this detailed examination of the test results, we hope to gain a better understanding of how these programming languages can be optimized and utilized in various contexts.

Table 4.1 Comprehensive Analysis of the Results

Test #	Language	Sources	Score	Time(seconds)
Test 1	Java	2	1.0	0.30s
Test 2	Java	2	0.71	0.38s
Test 1	Kotlin	2	0.49	0.36s
Test 2	Kotlin	2	1.0	0.41s
Test 1	CSHARP	2	1.0	0.29s
Test 2	CSHARP	2	0.0	0.33s
Test 1	PYTHON	2	0.0	0.38s
Test 2	PYTHON	2	1.0	0.40s
Test 1	C++	2	1.0	0.31s
Test 2	C++	2	0.62	0.37s

The results depicted in a Table 1, which is now being visually represented below. This means that the data has been organized and displayed in a clear and concise manner for easy understanding. Through this visual representation, it will be easier to analyse the data and draw conclusions from it. The use of tables and graphs is an effective way to present complex information in a simplified manner, making it accessible to a wider audience. Therefore, by presenting the test results in both tabular and visual forms, it allows for better comprehension and interpretation of the data.

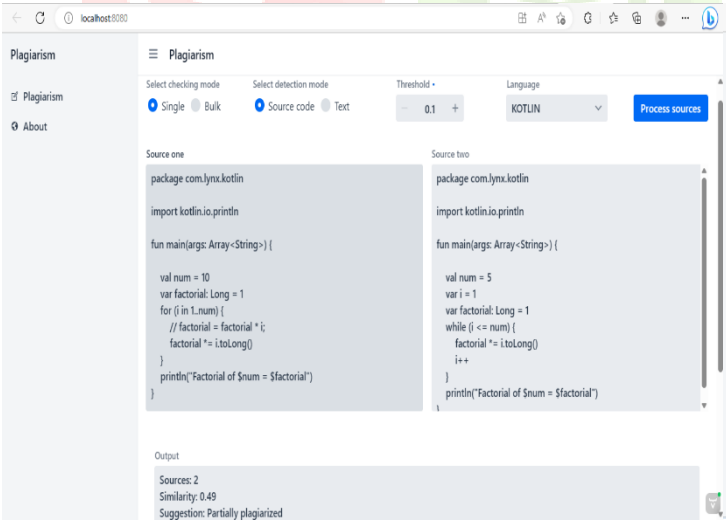


Figure 5: Kotlin test 1

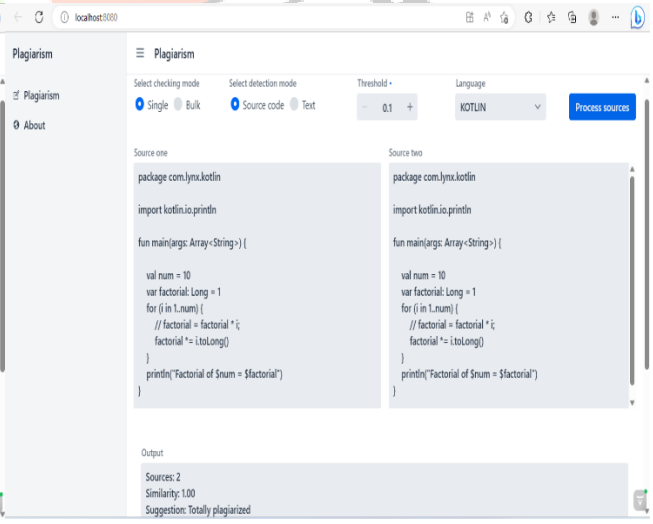


Figure 6: Kotlin test 2

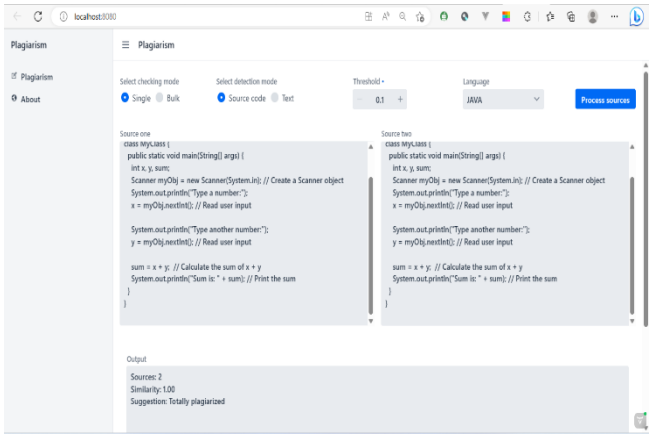


Figure 7: Java test

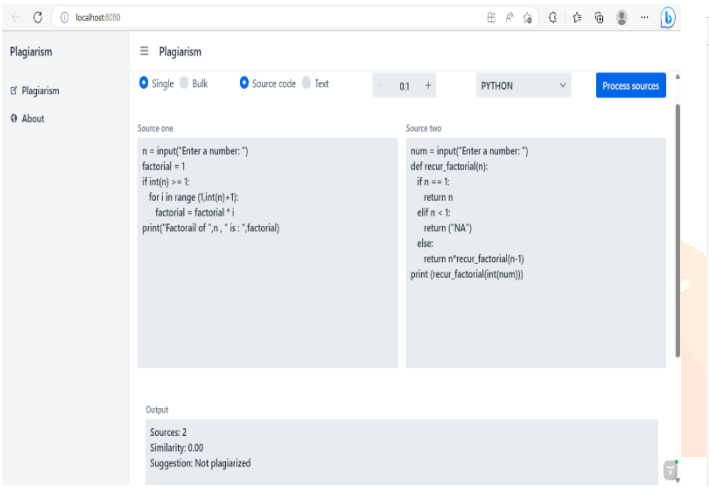


Figure 8: Python test 1

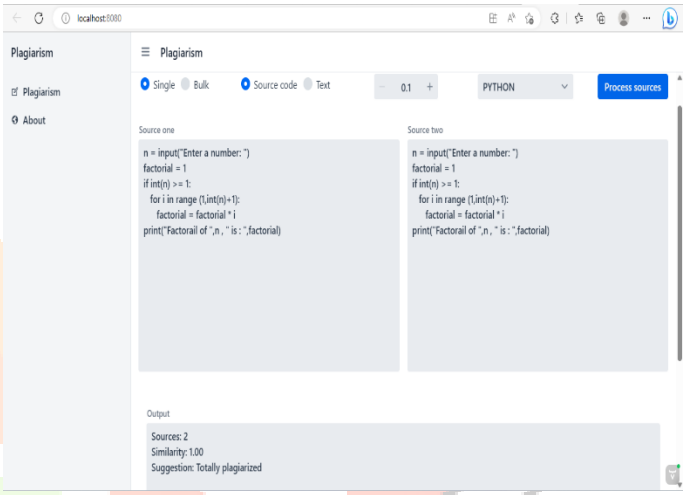


Figure 9: Python test 2

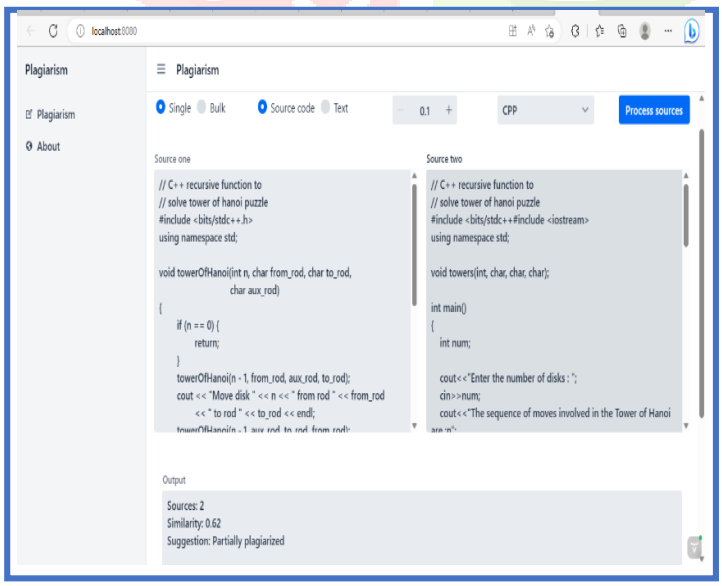


Figure 10: CSHARP test 1

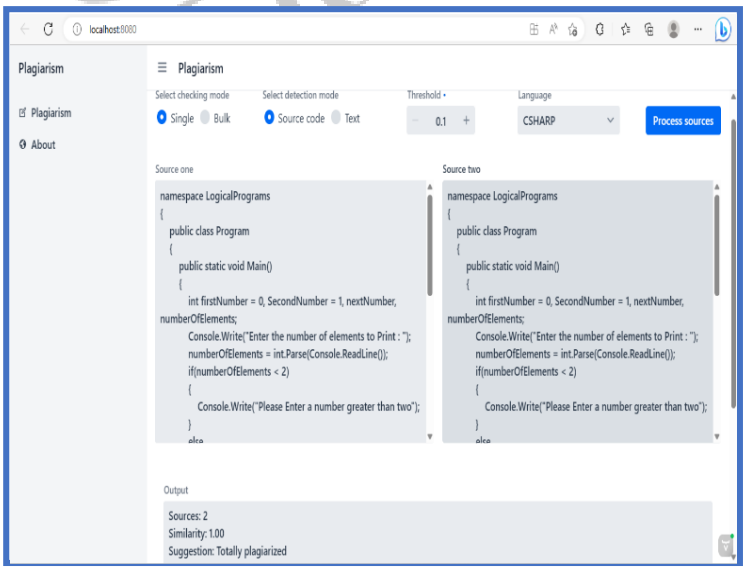


Figure 12: C++ test 1

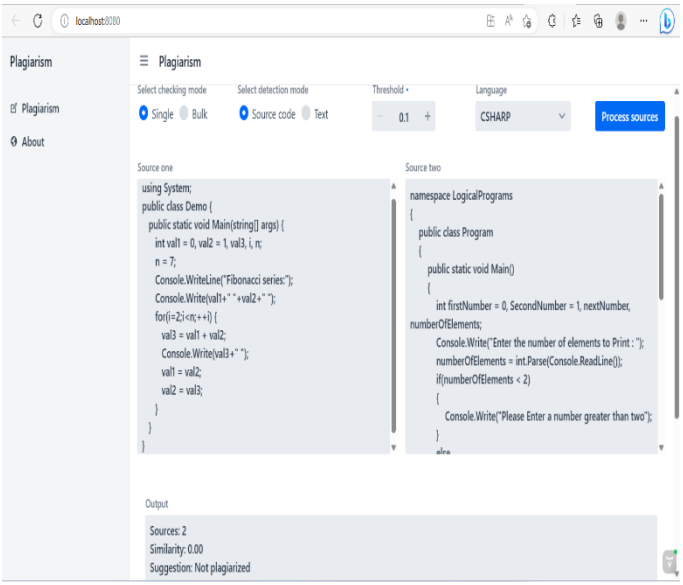


Figure 11: CSHARP test 2

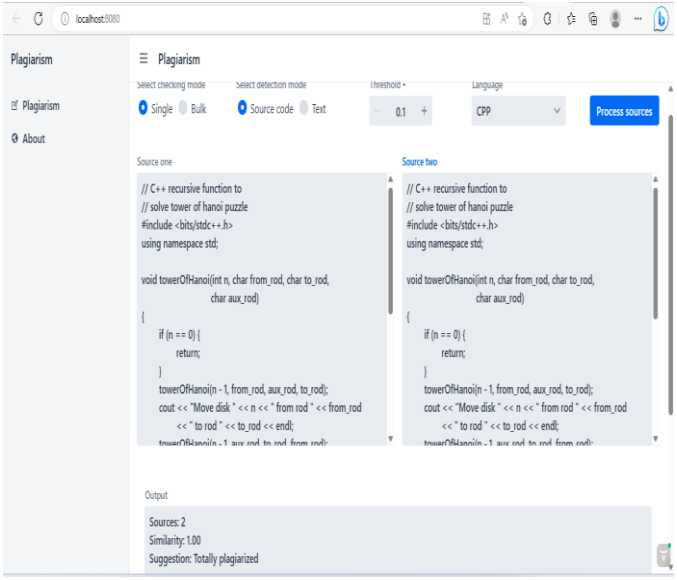


Figure 12: C++ test 1

Table 4.2

The following is the comparison result for all the

Test #	Language	Sources	Score	Time(s)	Suggestion
Test 1	Java	2	1.0	0.30s	
Test 2	Java	2	0.71	0.38s	
Test 1	Kotlin	2	0.49	0.36s	
Test 2	Kotlin	2	1.0	0.41s	
Test 1	CSHARP	2	1.0	0.29s	
Test 2	CSHARP	2	0.0	0.33s	
Test 1	PYTHON	2	0.0	0.38s	
Test 2	PYTHON	2	1.0	0.40s	
Test 1	C++	2	1.0	0.31s	
Test 2	C++	2	0.62	0.37s	

Table 4.2

The following is the comparison result for all the programming languages used

Prediction	TP	TN	FP	FN
Count	98	64	2	15

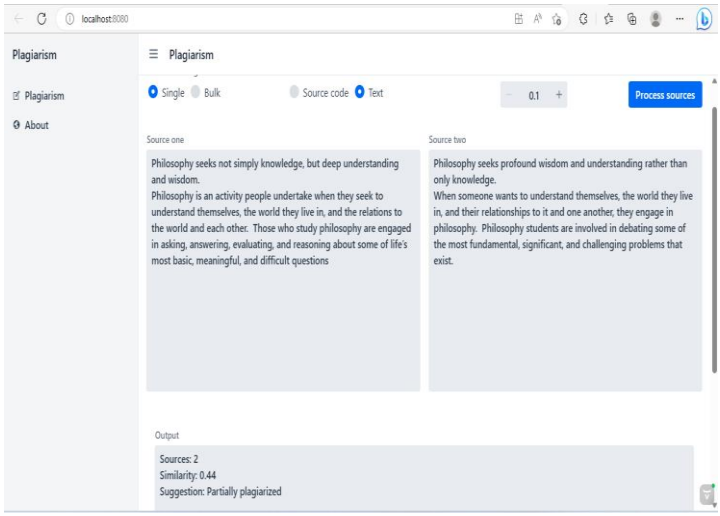


Figure 15: text comparison 2

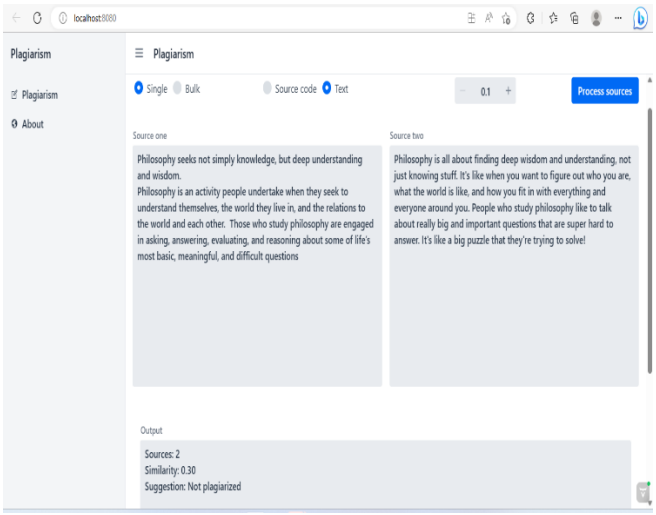


Figure 16: Test Comparison 3

4.2 Bulk Mode Plagiarism Detection

The detection of plagiarism can be achieved through different methods, one of which is the Bulk method. This approach involves users uploading compressed source code files in zip format for analysis. The interface provided for this purpose allows users to initiate the process easily and conveniently. Unlike the Single Mode 2's source code comparison, the Bulk method identifies instances of plagiarism and groups them together based on a predetermined minimum threshold. However, it is important to note that users cannot specify this threshold via the interface at present. To ensure accurate results, users must also select the language of their uploaded sources. This allows the system to process them appropriately using distinct Abstract Syntax Tree processing requirements for each programming language. Overall, the Bulk method offers an efficient way to detect plagiarism in large sets of source code files while maintaining high levels of accuracy and reliability.

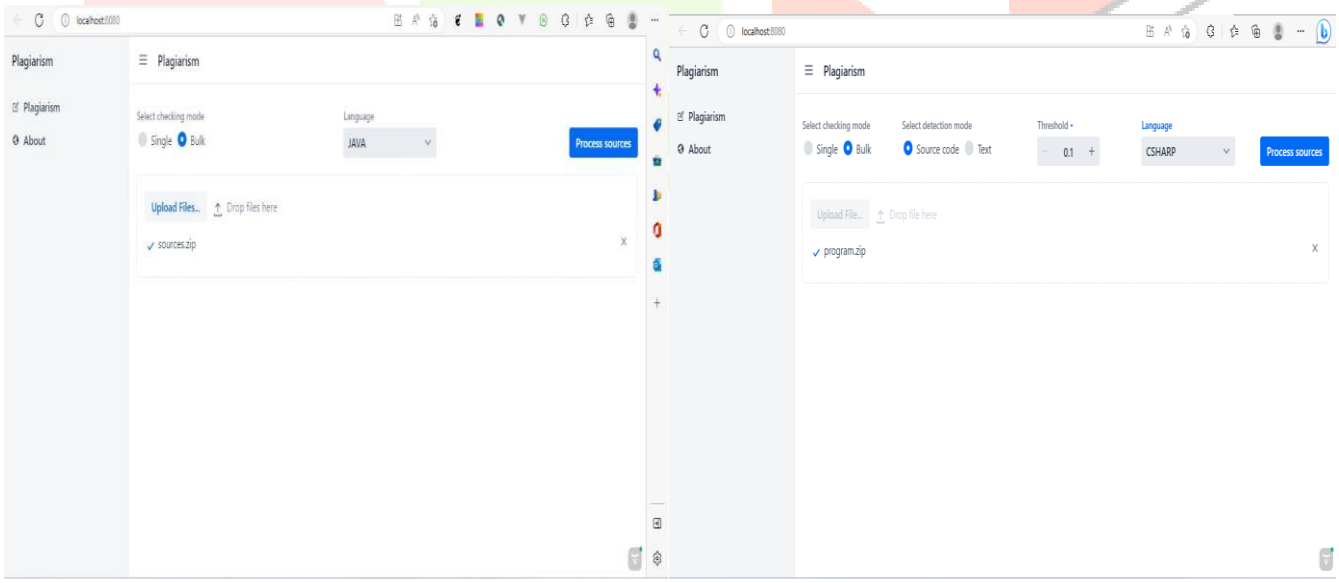


Figure 16: Bulk Mode Plagiarism Detection

Figure 17: Interface for Uploading Sources

4.3 Bulk Comparison results

The outcome of the comparison operation for programs coded in CSharp, involving 7 sources, is presented below. The process was completed within a span of approximately 2 to 3 seconds. Notably, two sources (i.e., second and third) encountered parsing errors and were thus excluded from further analysis; only the first, fourth, fifth, sixth and seventh sources remained for comparison purposes. Parsing failure occurs when the syntax employed is incorrect.

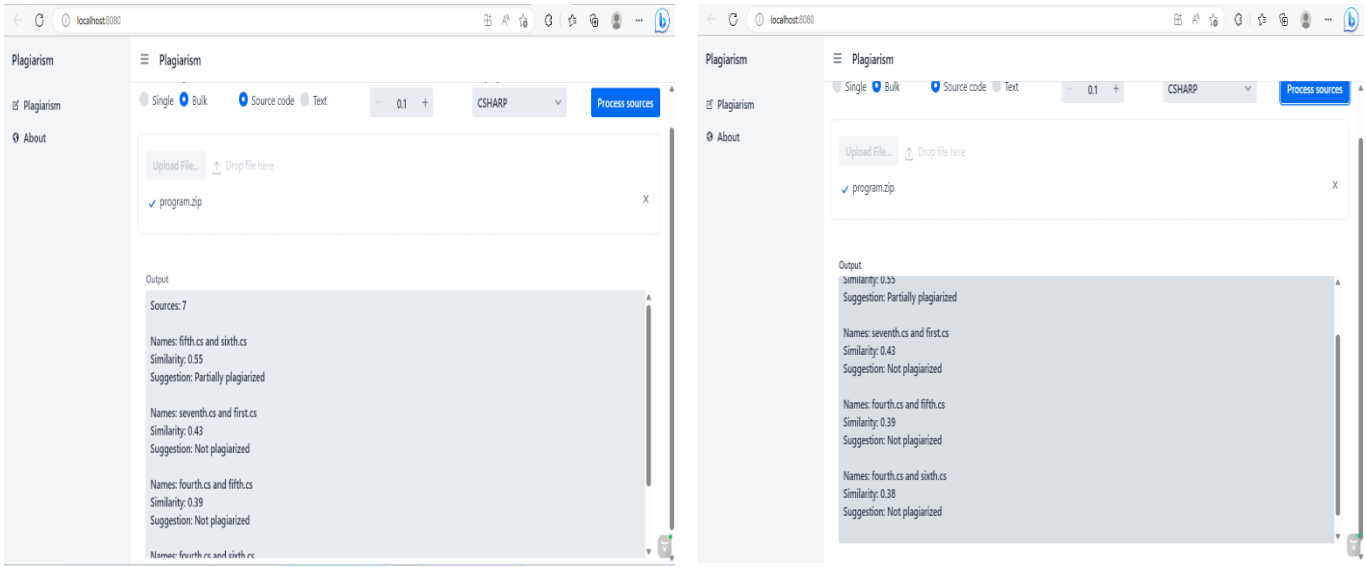


Figure 18 : Sources code output display

V. CONCLUSION

The purpose of the study is to propose a novel method for detecting code similarity among different programs. The method utilizes an abstract syntax implementation structure diagram, which has demonstrated promising results in effectively identifying instances of plagiarism. Compared to existing detection methods, the approach employed in this study has been found to be more accurate and precise. The process involves feature quantization calculation and translation into an abstract implementation structure diagram. Although these steps are complex, the study suggests that further exploration and refinement could enhance the method's effectiveness. The experimental results presented in the paper showcase the potential of this technique in identifying plagiarism while minimizing errors. As a result, the proposed method holds significant value as a tool for software developers and educators alike, facilitating better plagiarism detection and promoting academic integrity in programming and software development environments.

VI. RECOMMENDATION

The study introduces a novel method for detecting code similarity in different programs using an abstract syntax implementation structure diagram. The approach shows promise in effectively identifying plagiarism and outperforms existing detection methods in accuracy and precision. Further exploration is recommended to improve the method's effectiveness, especially in handling code obfuscation and variations. The study's experimental results demonstrate its potential as a valuable tool for software developers and educators in detecting plagiarism while minimizing errors. To strengthen its findings, extensive evaluation and benchmarking against real-world scenarios are suggested. Optimizing performance and addressing potential limitations would enhance the method's applicability and credibility.

REFERENCE

Adebayo, I., Adeyemi, I., & Onumanyi, P. (2017). Design and

Abuhamad, M., Jung, C., Mohaisen, D., & Nyang, D. (2023). SHIELD: Thwarting Code Authorship Attribution. *arXiv preprint arXiv:2304.13255*.

Asghari, H., Mohtaj, S., Fatemi, O., Faili, H., Rosso, P., & Potthast, M. (2018). Algorithms and corpora for persian plagiarism detection: overview of PAN at FIRE 2016. In *Text Processing: FIRE 2016 International Workshop, Kolkata, India, December 7–10, 2016, Revised Selected Papers* (pp. 61-79). Springer International Publishing.

Austin, M. J., & Brown, L. D. (1999). Internet plagiarism: Developing strategies to curb student academic dishonesty. *The Internet and higher education*, *2*(1), 21-33.

Barrón-Cedeño, A., Potthast, M., Rosso, P., Stein, B., & Eiselt, A. (2010, May). Corpus and Evaluation Measures for Automatic Plagiarism Detection. In *LREC*.

Chae, D. K., Ha, J., Kim, S. W., Kang, B., & Im, E. G. (2013, October). Software plagiarism detection: a graph-based approach. In *Proceedings of the 22nd ACM international conference on Information & Knowledge Management* (pp. 1577-1580).

Cooper, K. D., & Torczon, L. (2012). Chapter 5-intermediate representations. *Engineering a Compiler (Second Edition). Ed. by Keith D. Cooper and Linda Torczon. Second Edition. Boston: Morgan Kaufmann*, 221-268.

Duracik, M., Hrkut, P., Krsak, E. and Toth, S. (2020): Abstract Syntax tree based code Antiplagiarism System for large Project set. *IEEE Access* Digital Object Identifier 10.1109/ACCESS.2020.3026422

Ison, D. C. (2015). The influence of the Internet on plagiarism among doctoral dissertations: An empirical study. *Journal of Academic Ethics*, *13*, 151-166.

Maurer, H. A., Kappe, F., & Zaka, B. (2006). Plagiarism-A survey. *J. Univers. Comput. Sci.*, *12*(8), 1050-1084.

Neamtiu, I., Foster, J. S., & Hicks, M. (2005, May). Understanding source code evolution using abstract syntax tree matching. In *Proceedings of the 2005 international workshop on Mining software repositories* (pp. 1-5).

Shrestha, P., & Solorio, T. (2013). Using a Variety of n-Grams for the Detection of Different Kinds of Plagiarism. *Notebook for PAN at CLEF*, *2013*.

Torres, S., & Gelbukh, A. (2009). Comparing similarity measures for original WSD lesk algorithm. *Research in Computing Science*, *43*, 155-166.

Ullah, F., Jabbar, S., & Al-Turjman, F. (2020). Programmers' de-anonymization using a hybrid approach of abstract syntax tree and deep learning. Technological Forecasting and Social Change, 159, 120186.

Ullah, F., Wang, J., Jabbar, S., Al-Turjman, F., & Alazab, M. (2019). Source code authorship attribution using hybrid approach of program dependence graph and deep learning model. *IEEE Access*, *7*, 141987-141999.