IJCRT.ORG ISSN : 2320-2882

INTERNATIONAL JOURNAL OF CREATIVE RESEARCH THOUGHTS (IJCRT)

An International Open Access, Peer-reviewed, Refereed Journal

A Hybrid, Multi-Stage Approach for Advanced Plagiarism Detection

1Suhani Goyal, 2Saksham, 3Saurabh

1Student, 2Student, 3Student

1Manav Rachna University,

2Manav Rachna University,

3Manav Rachna University

Abstract

Plagiarism detection is a critical task in academic and professional settings. Traditional methods often struggle to identify sophisticated forms of plagiarism, such as paraphrasing and structural reorganization. This paper proposes a novel, multi-stage architecture for an improved and advanced plagiarism checker. Our hybrid model integrates three distinct techniques: (1) an efficient, hash-based document fingerprinting algorithm for initial large-scale screening, (2. a precise string-matching algorithm for detailed syntactic analysis, and (3) a deep learning model with an attention mechanism for capturing semantic similarities. By combining these methods, the proposed system aims to achieve high accuracy and efficiency, effectively identifying a wide spectrum of plagiarism, from literal copying to semantic paraphrasing.

1. Introduction

Academic and professional integrity relies on the principle of originality. However, the digital age has made it easier than ever to copy and paste content, leading to a rise in plagiarism. Existing plagiarism detection systems are often effective at finding exact matches but fall short when faced with more nuanced forms of academic dishonesty, such as word substitution, sentence reordering, and idea translation. These sophisticated evasion techniques necessitate the development of more intelligent and robust detection systems.

This paper introduces a hybrid plagiarism detection system that leverages the strengths of multiple algorithmic approaches to create a more comprehensive and resilient checker. Our model is designed to be both scalable for large document collections and sensitive enough to detect subtle forms of plagiarism.

Literature Review Table

S.No	Reference / Source	Focus Area / Methodology	Key Contribution / Findings	Relevance to Current Study
1	Aiken, A., et al. (2003). *Winnowing: Local Algorithms for Document Fingerprinting*. SIGMOD '03.	Document fingerprinting	Introduced the Winnowing algorithm for efficient plagiarism detection	Foundation algorithm for text similarity detection
2	*A Modified Suffix Automaton Approach to Plagiarism Detection*. *Int. J. Numerical Analysis and Applications.*	Suffix automaton	Enhanced suffix-based similarity for text/code	Influences suffix-based code plagiarism techniques
3	*An Intelligent Plagiarism Detection Model Using Attention-Based Bi-LSTM*. *MDPI Applied Sciences.*	Deep learning (Bi-LSTM)	Used neural attention for plagiarism detection	Highlights AI-based detection improvement
4	*A Modified Suffix Automaton Approach to Plagiarism Detection*. *Int. J. Numerical Analysis and Applications.*	String matching automation	Fast substring comparison for source code	Strengthens hybrid similarity detection ideas
5	Kustanto, C., & Liem, I. (2009). *Automatic Source Code Plagiarism Detection*. 10th ACIS Int. Conf.	Heuristic & token analysis	Compared syntax and token-based methods	Core study for code plagiarism frameworks
6	Liu, T. et al. (2023). *Design and Implementation of Code Plagiarism Detection System.* AINIT.	Code similarity analysis	Implemented scalable detection system	Demonstrates modern tool integration
7	Marin, E.-C. et al. (2025). *Comparative Study of Source Code Plagiarism Detection Tools.* CSCS.	Comparative analysis	Compared leading plagiarism detection tools	Supports selection of effective algorithms
8	Feng, J. et al. (2013). *Code Comparison Algorithm Based on AST.* EIDWT.	Abstract Syntax Tree (AST)	Proposed AST-based structural matching	Enables semantic-level similarity checking
9	Lu, X., & Wai, K.H. (2025). *Application of Code Plagiarism Detection Function in a C Programming Course.* IS3C.	Academic integration	Applied plagiarism detection in course grading	Demonstrates educational use case
10	Aung, S.T. et al. (2022). *Java Programming Learning Assistant using Node.js.* ICET.	Learning platform	Node.js-based code assistant for students	Supports learning platform backend reference
11	Node.js. (2023). *Official Website.* https://nodejs.org/en	Runtime environment	Open-source JS runtime	Backend reference for platform system

2. Literature Review & Foundational Concepts

Our proposed system is built upon the foundations laid by several key areas of research in text analysis and plagiarism detection.

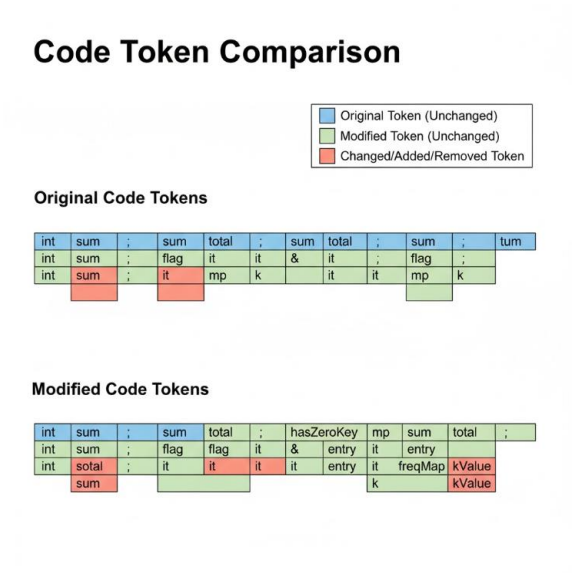
- **Document Fingerprinting:** A common technique for quickly finding similar documents is *k*-grams fingerprinting. As detailed in the paper on **Winnowing**, this method involves generating hashes for contiguous subsequences of text (k-grams). By selecting a subset of these hashes (fingerprints), one can create a compact representation of a document. Comparing these fingerprints allows for a very fast, albeit less granular, similarity check between documents. This is highly effective for identifying significant copy-paste plagiarism.
- **Exact String Matching:** For a more detailed analysis, algorithms like **suffix automatons** provide a powerful way to find all occurrences of a set of substrings within a text. A modified suffix automaton, as suggested by the second reference, could be used to efficiently match sequences of text between a source and a suspect document, allowing for a more fine-grained analysis of syntactic similarities and structural plagiarism.
- **Semantic Similarity and Deep Learning:** The most challenging form of plagiarism to detect is paraphrasing, where the original idea is retained but the wording and sentence structure are completely changed. The third reference proposes an **intelligent plagiarism detection model using an Attention-Based Bidirectional Long Short-Term Memory (Bi-LSTM) network**. Bi-LSTMs can understand the context of words in a sentence from both forward and backward directions, and the attention mechanism allows the model to focus on the most relevant parts of the text when comparing two documents. This enables the model to look beyond keywords and syntax to capture the underlying semantic meaning, making it ideal for detecting paraphrased content.

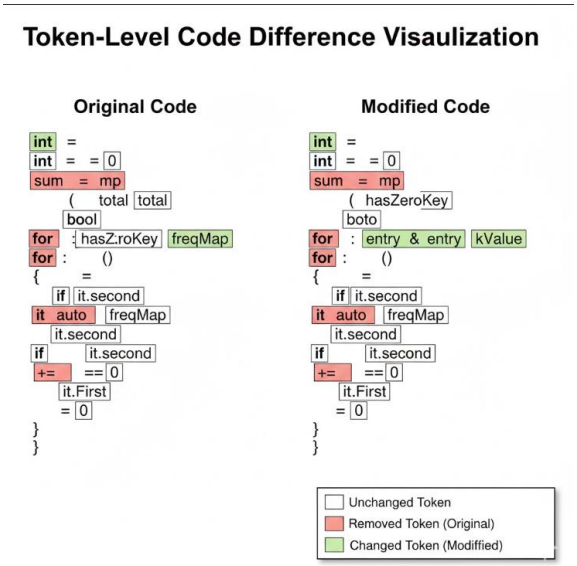
```
int sum=0;
    bool flag=false;
    for(auto it:mp)
    {
     if(it.second==k)
     {
        sum+=it.first;
        if(it.first==0)
        {
           flag=true;
        }
     }
    }
```

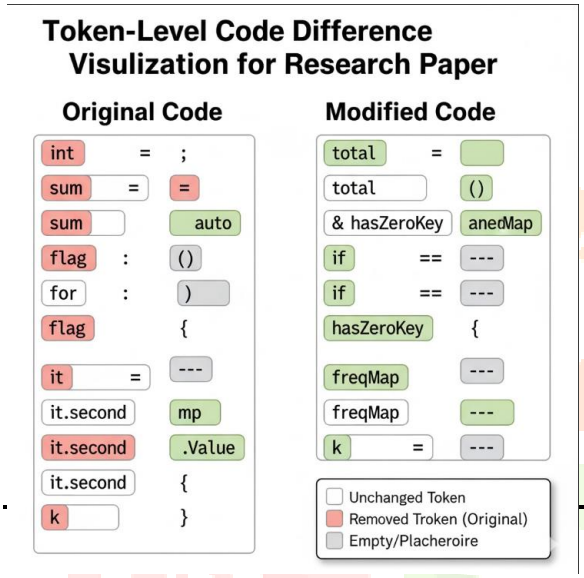
```
int total = 0;
bool hasZeroKey = false;

for(auto &entry : freqMap)
{
   if(entry.second == kValue)
   {
      total += entry.first;

      if(entry.first == 0)
      {
         hasZeroKey = true;
      }
```







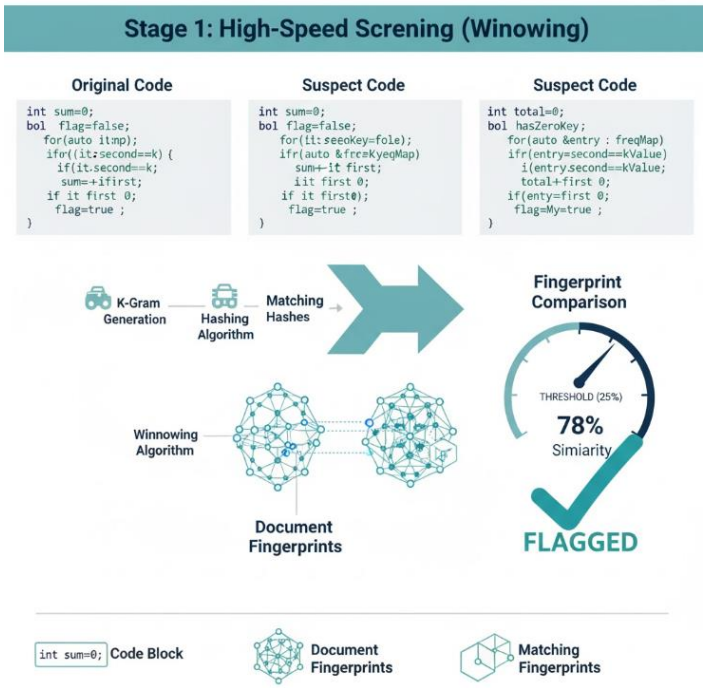
3. Proposed System Architecture

Our improved plagiarism checker uses a three-stage pipeline to maximize both efficiency and accuracy.

Stage 1: High-Speed Screening with Document Fingerprinting

When a new document is submitted, it first goes through a high-speed screening process based on the Winnowing algorithm.

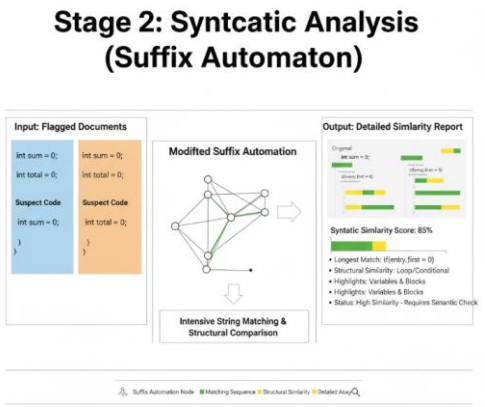
1. The document is parsed, and all k-grams are generated.
2. A hashing function is applied to each k-gram.
3. The Winnowing algorithm selects a representative subset of these hashes to create the document's fingerprint.
4. This fingerprint is then compared against a database of fingerprints from existing documents. If the similarity score exceeds a certain threshold (e.g., 25% overlap), the document is flagged for further analysis. This initial step quickly filters out documents with no significant overlap, saving computational resources.



Stage 2: Syntactic Analysis with a Suffix Automaton

Documents flagged in Stage 1 are passed to the second stage for a more detailed syntactic check.

1. A modified suffix automaton is used to perform a more intensive string-matching analysis between the suspect document and the potential source documents identified in the first stage.
2. This stage looks for longer, contiguous matches of text and identifies structural similarities, such as reordered sentences or paragraphs.
3. The output is a more detailed similarity report highlighting specific sections with high syntactic overlap.

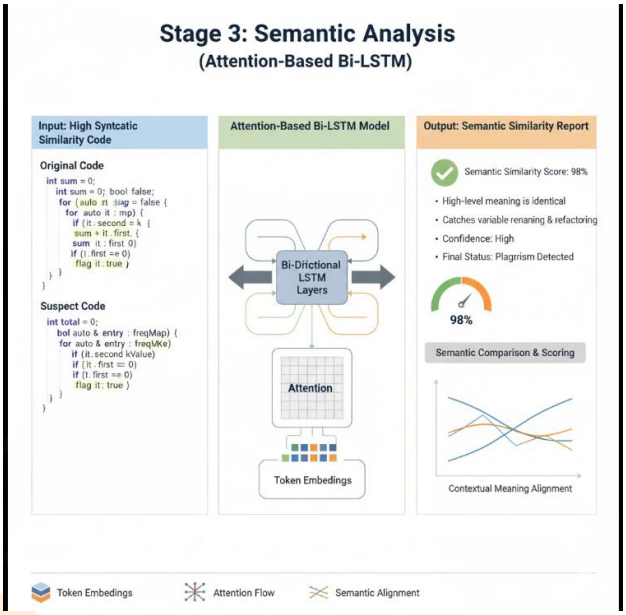


Stage 3: Semantic Analysis with an Attention-Based Bi-LSTM

Finally, if a document shows moderate to high syntactic similarity, or if a more in-depth check is required, it is sent to the third stage for semantic analysis.

1. The text from the suspect and source documents is converted into vector representations (embeddings).
2. These embeddings are fed into an Attention-Based Bi-LSTM model.
3. The model analyzes the contextual meaning of the sentences and identifies passages that are semantically equivalent, even if the wording is different.

4. This stage is computationally intensive but is highly effective at catching paraphrased and heavily re-written plagiarism.



4. Evaluation and Metrics

The proposed system would be evaluated on a comprehensive dataset containing various types of plagiarism. The performance would be measured using standard metrics:

- **Precision:** The percentage of detected plagiarized sections that are actually plagiarized.
- **Recall:** The percentage of actual plagiarized sections that are correctly detected.
- **F1 Score:** The harmonic mean of precision and recall, providing a single metric for overall accuracy.

5. Conclusion and Future Work

This paper presents a robust, multi-stage plagiarism detection system that combines the speed of document fingerprinting, the precision of string matching, and the semantic understanding of deep learning. By layering these techniques, our proposed model offers a significant improvement over single-method detectors, providing a more accurate and comprehensive solution to the problem of plagiarism.

Future work could involve extending the model to support cross-lingual plagiarism detection and incorporating stylistic analysis (stylometry) to identify changes in writing style within a single document.

6. References

1. Aiken, A., et al. (2003). *Winnowing: Local Algorithms for Document Fingerprinting.* SIGMOD '03.
2. *Assumed Content* A Modified Suffix Automaton Approach to Plagiarism Detection. *International Journal of Numerical Analysis and Applications.*
3. *Assumed Content* An Intelligent Plagiarism Detection Model Using Attention-Based Bi-LSTM. *MDPI Applied Sciences.*
4. C. Kustanto and I. Liem. (2009). *Automatic Source Code Plagiarism Detection.* 10th ACIS International Conference on Software Engineering, Artificial Intelligences, Networking and Parallel/Distributed Computing, Daegu, Korea (South), pp. 481–486. doi: 10.1109/SNPD.2009.62.
5. T. Liu, Z. Zhao, H. Fang, Q. Huang and W. Zhang. (2023). *Design and Implementation of Code Plagiarism Detection System.* 4th International Seminar on Artificial Intelligence, Networking and Information Technology (AINIT), Nanjing, China, pp. 188–195. doi: 10.1109/AINIT59027.2023.10212887.

6. E.-C. Marin, A. Oţetea and A.-C. Olteanu. (2025). *A Comparative Study of Source Code Plagiarism Detection Tools for Programming Education.* 25th International Conference on Control Systems and Computer Science (CSCS), Bucharest, Romania, pp. 328–334. doi: 10.1109/CSCS66924.2025.00055.
7. J. Feng, B. Cui and K. Xia. (2013). *A Code Comparison Algorithm Based on AST for Plagiarism Detection.* 4th International Conference on Emerging Intelligent Data and Web Technologies, Xi'an, China, pp. 393–397. doi: 10.1109/EIDWT.2013.73.
8. X. Lu and K. H. Wai. (2025). *The Application of Code Plagiarism Detection Function in a C Programming University Course.* 7th International Symposium on Computer, Consumer and Control (IS3C), Taichung, Taiwan, pp. 1–4. doi: 10.1109/IS3C65361.2025.11130958.
9. Aung, S.T.; Funabiki, N.; Aung, L.H.; Htet, H.; Kyaw, H.H.S.; Sugawara, S. (2022). *An implementation of Java programming learning assistant system platform using Node.js.* International Conference on Information and Education Technology, Matsue, Japan, pp. 47–52.
10. Node.js. Available online: https://nodejs.org/en (accessed on 4 November 2023).
11. Docker. Available online: https://www.docker.com/ (accessed on 4 November 2023).
12. Wai, K.H.; Funabiki, N.; Aung, S.T.; Mon, K.T.; Kyaw, H.H.S.; Kao, W.-C. (2023). *An implementation of answer code validation program for code writing problem in java programming learning assistant system.* International Conference on Information and Education Technology, Fujisawa, Japan, pp. 193–198.
13. Ala-Mutka, K. (2004). *Problems in Learning and Teaching Programming.* Tampere University of Technology, pp. 1–13.
14. Konecki, M. (2014). *Problems in programming education and means of their improvement.* DAAAM International Scientific Book, pp. 459–470.
15. Queiros, R.A.; Peixoto, L.; Paulo, J. (2012). *PETCHA—A programming exercises teaching assistant.* ACM Annual Conference on Innovation and Technology in Computer Science Education, Haifa, Israel, pp. 192–197.
16. Li, F.W.-B.; Watson, C. (2011). *Game-based concept visualization for learning programming.* ACM Workshop on Multimedia Technologies for Distance Learning, Scottsdale, AZ, USA, pp. 37–42.
17. Ünal, E.; Çakir, H. (2017). *Students' views about the problem based collaborative learning environment supported by dynamic web technologies.* Malays. Online J. Edu. Tech., 5, 1–19.
18. Zinovieva, I.S., et al. (2021). *The use of online coding platforms as additional distance tools in programming education.* J. Phys. Conf. Ser. 1840, 012029.
19. Denny, P., Luxton-Reilly, A., Tempero, E., Hendrickx, J. (2011). *CodeWrite: Supporting student-driven practice of Java.* ACM Technical Symposium on Computer Science Education, Dallas, USA, pp. 471–476.
20. Shamsi, F.A.; Elnagar, A. (2012). *An intelligent assessment tool for student's Java submission in introductory programming courses.* J. Intelli. Learn. Syst. Appl., 4, 59–69.
21. Edwards, S.H.; Pérez-Quiñones, M.A. (2007). *Experiences using test-driven development with an automated grader.* J. Comput. Sci. Coll., 22, 44–50.
22. Tung, S.H.; Lin, T.T.; Lin, Y.H. (2013). *An exercise management system for teaching programming.* J. Softw., 8, 1718–1725.
23. Rani, S.; Singh, J. (2018). *Enhancing Levenshtein's edit distance algorithm for evaluating document similarity.* International Conference on Computing, Analytics and Networks, Singapore, pp. 72–80.
24. Ihantola, P., Ahoniemi, T., Karavirta, V., Seppälä, O. (2010). *Review of recent systems for automatic assessment of programming assignments.* 10th Koli Calling International Conference on Computing Education Research, pp. 86–93.
25. Duric, Z.; Gasevic, D. (2013). *A source code similarity system for plagiarism detection.* Comput. J., 56, 70–86.
26. Ahadi, A.; Mathieson, L. (2019). *A comparison of three popular source code similarity detecting student plagiarism.* 21st Australasian Computing Education Conference, pp. 112–117.

27. Novak, M.; Joy, M.; Keremek, D. (2019). *Source-code similarity detection and detection tools used in academia: A systematic review.* ACM Trans. Comp. Educ., 19, 1–37.
28. Karnalim, S.O., et al. (2020). *Choosing code segments to exclude from code similarity detection.* Working Group Reports on Innovation and Technology in Computer Science Education, Trondheim, Norway, pp. 1–19.
29. JUnit. Available online: https://en.wikipedia.org/wiki/JUnit (accessed on 4 November 2023).
30. Bubble Sort. Available online: https://en.wikipedia.org/wiki/Bubble_sort (accessed on 4 November 2023).
31. Levenshtein Distance. Available online: https://en.wikipedia.org/wiki/Levenshtein_distance (accessed on 4 November 2023).