Indonesian Journal of Electrical Engineering and Computer Science
Vol. 38, No. 3, June 2025, pp. 1924~1935
ISSN: 2502-4752, DOI: 10.11591/ijeecs.v38.i3.pp1924-1935

Plagiarism detection in Verilog and textual content using linguistic features

Kirankumar Manivannan[1], Nalayini C. M.[2], Sathya V.[3], Kumar G.[4], Dinesh Babu M.[5]
[1]School of Electronics Engineering, Vellore Institute of Technology, Chennai Campus, Tamil Nadu, India
[2]Department of Information Technology, Velammal Engineering College, Chennai, Tamil Nadu, India
[3]School of Computing, Vel Tech Rangarajan Dr. Sagunthala R&D Institute of Science and Technology, Chennai, India
[4]Faculty of Management, SRM Institute of Science and Technology, Kattankulathur, Tamil Nadu, India
[5]Department of Mechanical Engineering, Rajalakshmi Institute of Technology, Chennai, Tamil Nadu, India

Article Info

Article history:

Received Jun 20, 2024
Revised Dec 10, 2024
Accepted Feb 27, 2025

Keywords:

Linguistic characteristics
Plagiarism
Programming code theft
Verilog algorithms
WordNet lexical database

ABSTRACT

The illicit act of appropriating programming code has long been an appealing notion due to the immediate time and effort savings it affords perpetrators. However, it is universally acknowledged that concerted efforts are imperative to identify and rectify such transgressions. This is particularly crucial as academic institutions, including universities, may inadvertently confer degrees for work tainted by this form of plagiarism. Consequently, the primary objective of this research is to scrutinize the feasibility of identifying plagiarism within pairs of Verilog algorithms and texts. this study aims to detect plagiarism in textual content and Verilog code by leveraging diverse linguistic characteristics from the WordNet lexical database. The primary objective is to achieve optimal accuracy in identifying instances of plagiarism, incorporating features such as modifications to text structure, synonym substitution, and simultaneous application of these strategies. The system's architecture is intricately designed to unveil instances of plagiarism in both textual content and Verilog code by extracting nuanced characteristics. The systematic process includes preprocessing, detailed analysis, and post-processing, supported by a feature-rich database. Each entry in the database represents a distinctive similarity case, contributing to a thorough and comprehensive approach to plagiarism detection.

Corresponding Author:

Sathya V.
Department of Computer Science and Engineering, Vel Tech Rangarajan Dr. Sagunthala R&D Institute of Science and Technology
Chennai, Tamil Nadu India
Email: saro.sath@gmail.com

1. INTRODUCTION

It's reasonable to argue that there has never been more room for plagiarism in programming code. There doesn't appear to be a need to write code from scratch anymore, what with the proliferation of information available from sources like the Internet. For educational institutions, this has caused more than a headache because their goal is to grant degrees to individuals who have attained a specific level of mental ability rather than just those who have received passing grades. Many individual students view programming classes as a means of obtaining respectable outcomes with the least amount of work. This occurs frequently because there is a strong temptation to just copy code from other sources and pass it off as original.

Therefore, the identification of plagiarism between pairs of Verilog algorithms-for example, algorithms turned in for a university assignment-is taken into consideration in this study. It is fairly tricky even impractical, to track down as to demonstrate that descriptions are being altered in any way from outside sources, but it is very feasible to demonstrate that two students have collaborated. This reveals the primary objective, which is to be able to determine with clarity whether a particular piece of coding has been inherited in any way from another. The project's current restriction is that it will only search Verilog code for instances of plagiarism. Time restraints and the awareness that the language is extensively used and respected—particularly in educational institutions-are the two reasons for this. The aforementioned goal will be partially attained by compiling a data set. It will contain a variety of Verilog methods, some of which are original and some of which are true copying from other sources. This set will serve as the source of samples for the test set in addition to enabling the calculation of thresholds for the selected detection techniques. The resultant figure will show plagiarism if it is exceeded or, in some situations, if it is not. The purpose of this research is to identify plagiarism, so understanding the differences between attribute counting systems and structural metric systems is essential. It is important to note that attribute counting systems, as opposed to structural metric systems, only measure the extent to which a given attribute is present inside a given code length.

Plagiarism of source codes, defined as the unauthorized copying or imitation of another pupil's code without appropriate citation, has become a common phenomenon especially within academic setting especially in computer science and engineering disciplines whose programming assignments forms a significant portion of the course work [1], [2]. The rise in various online code repositories availability and collaboration among students and others in group projects, despite the effectiveness they bring in other ways, has continued to nurture the act of plagiarism, posing a great threat to academic honesty [3]. In order to maintain the level of education and, therefore, avoid making biased decisions, there is a need to utilize research methods aimed at identifying source code plagiarism [4]. Many solutions have been suggested and applied so that this issue can be solved effectively, although some of them are relevant and effective in some ways while others are not [4]-[6]. They span from simple textual comparisons vulnerable to pro-grammers' attempts to 'fool' the diff tool by making tiny code changes to more complex ones that operate at the structural and stylistic levels [7]. Some of the researchers have used the principles and concepts of machine learning to categorize and recognize the similarities in the code submissions [8], while others have employed the rules defined by the experts and including heuristics [9]. The appendix further illuminates that the method's challenges are compounded by the diversity in programming languages and the paradigms taught in academic courses. For example, VHDL (VHSIC hardware description language), which is the domain-specific language used for digital circuits description and simulation, has its syntax and structure that differ from real high-level languages used in OS development such as C ++, Java, or Python [10]. This mandates for advanced anti plagiarism software's that are 'multifaceted' in the sense that they should be able to effectively cope up with a wide variety of coding languages as well as coding paradigms.

Positive findings regarding source code plagiarism prevalence and type in academic settings have emerged from several scholarly studies. Similar to [1], reported that a large number of students confessed to engaging internet resources for submitting programming assignments and some go to the extent of copying solutions in their totality [1]. This underscores the need for more protective educational measures that would enable cultivation of ethical principles among the students while at the same time deeming it relevant and crucial for the students to engage in independent programs that would help in the acquisition of programming skills among others. On similar lines, in [2] used analogue-type clustering techniques to cluster similar group of code submissions that form potential clue of the plagiarism. This kind of approach illustrates the possible of applying the machine learning in the detection of the suspicious submissions in order to prevent instructors from spending time on these cases. Besides the usual method of using algorithms for detection of plagiarism, there have been some research into usage of anti-patterns and style analysis for detection of replicated code [5], [6]. These methodologies take advantage of the look and feel that is characteristically lodged to programmers whereby a certain programmer has a given coding style and preferred way of coding other than the rest. The employment of modern advanced language models like ChatGPT in creating content has further added a classical feature to confronting of plagiarism [11]. These models have the ability to produce what looks like genuine code based on a given natural language input, and this prospect has sparked anti-cheating sentiments among institution administrators with regards to student submission. As a result, authors are already trying to get deeper into finding unique approaches to distinguish AI-produced code from typical code written by developers.

Hence, much as there have been tremendous progress in the area of source code plagiarism detection, there are quite a number of hurdles and research queries that are yet to be addressed in the field. Different techniques of detection may work differently from one language to another and the problem has not been solved completely till date [12]. Present day research activities aimed at creating tools that are more resistant to diverse languages and paradigms [13]. Moreover, there are aspects that are ethical and pedagogical that must be considered as most of the detection technologies focus on education and the promotion of high standards of academic integrity [14]. The detection of plagiarism in Verilog code and associated technical documentation poses significant challenges for educational institutions and industry professionals. Traditional plagiarism detection methods often prove inadequate when applied to domain-specific languages like Verilog, failing to account for the unique characteristics of hardware description syntax. Furthermore, the concurrent occurrence of code and text plagiarism in technical projects necessitates an integrated approach capable of identifying similarities across diverse content types. The consequences of undetected plagiarism are severe, potentially compromising the integrity of academic research, intellectual property rights, and the overall quality of engineering education and practice. As such, there is an urgent need for more sophisticated, tailored approaches to plagiarism detection in this specialized field.

Thus, this research paper seeks to make a considerable addition to the existing research on efficient detection of source code plagiarism, by reviewing those techniques, outlining their advantages and shortcomings, and discussing potential future developments. This paper aims at providing an extensive review of the literature on the current state of affairs and arm educators, researchers, and developers with conceptual and practical tools to prevent the prevalence of dishonesty in programming courses.

According to the analysis the ubresolved problems and areas of improvement can be given as ;imited effectiveness in detecting sophisticated plagiarism techniques in Verilog code, such as structural reorganization and module renaming. Lack of integrated approaches that can simultaneously detect plagiarism in both code and textual content, crucial for comprehensive analysis of technical projects. Insufficient capability in identifying plagiarism involving synonym substitution and paraphrasing in technical documentation. Scalability issues in plagiarism detection methods for large Verilog projects and extensive technical documentation. Inadequate consideration of domain-specific features of hardware description languages in existing plagiarism detection tools.

To address these gaps, our research proposes a novel, integrated approach to plagiarism detection that combines linguistic features extracted from the WordNet lexical database with an adapted version of the greedy string tiling (GST) algorithm. Our approach offers several key innovations: a unified framework for detecting plagiarism in both Verilog code and associated textual content, enabling comprehensive analysis of technical projects. Utilization of linguistic features to improve the detection of synonym substitution and paraphrasing in technical writing. Adaptation of the GST algorithm to account for the specific syntax and structure of Verilog, enhancing its effectiveness for hardware description language plagiarism detection. Incorporation of scalable techniques to handle large-scale Verilog projects and extensive technical documentation. Integration of domain-specific knowledge of hardware description languages to improve detection accuracy and reduce false positives.

The remainder of this paper is structured as follows: section 2 presents a detailed review of related work, expanding on the contributions mentioned above and providing a comprehensive overview of the state of the art in plagiarism detection for programming languages and technical writing. Section 3 describes our proposed methodology in detail, including: the process of linguistic feature extraction from technical text. The adaptation of the GST algorithm for Verilog code. The integration of these components into a unified plagiarism detection framework. The incorporation of domain-specific knowledge for improved accuracy. Section 4 outlines our experimental setup, describing: the dataset collection process and characteristics. The metrics used for evaluation. The implementation details of our system. The baseline methods used for comparison. Section 5 presents the results of our experiments, providing: a comprehensive analysis of the performance of our proposed method compared to existing approaches. Detailed discussion of the implications of these results. Critical evaluation of the strengths and limitations of our approach. Section 5 concludes the paper by: Summarizing our key findings. Discussing the broader implications of our work for academic integrity and software development practices. Suggesting directions for future research in this field. Addressing potential ethical considerations and limitations of automated plagiarism detection.

2. RELATED WORK

Although the complete grammar of programming languages may be described and detailed[15] points out that the language of nature is far more complicated and imprecise, making it more difficult to create an effective copyright infringement detection system for programming languages. This helps to support the project's overarching goal of finding plagiarism exclusively in software code. Therefore, it was thought that this report was a good location to start the project, since it goes on to evaluate the several approaches that can be used to successfully detect software plagiarism. These include, to mention two of the attribute counting methods [16], and Halstead's software science metrics. He subsequently gives a demonstration of a simple method that uses structure metrics, or string comparisons, to identify plagiarism. In his extensive review of the available plagiarism tools, he clearly draws a distinction, pointing out that structural metric techniques are often seen as more likely to succeed than attribute calculating ones.

In his work on software "forensics," [17] limits the application of attribute counting approaches, which are analyzed to be more helpful in determining software authorship in software code than in detecting plagiarism in code. This is justified by stating that the measurements produce "the values which are definitely program-specific," which adds to the amount of evidence that supports the decision to overlook authorship attribution concerns in this project. More intriguingly, though, Sallis continues by identifying a "six-tuple vector" of features of computer code that, in his opinion, "should enable effective plagiarism detection." Verco and Wise conducted a crucial review in order to identify plagiarism They have made the bold move of contrasting the effectiveness of attribute counting and structural metric approaches, with intriguing findings. As indicated earlier on, the problem of plagiarism especially in the area of specialization that is computer science has been undertaken and explored comprehensively in several scripts and papers available in literature. Bretag *et al.* [17] described about the AST for identifying code clones which has the potential to be used for detecting the plagiarized code. This method is more beneficial in the identification of structural resemblance of the codes even when there is change in the names of variables or comments.

Park *et al.* [18], investigated how contract cheating is associated with the assessment's construction. Overall, based on their research, they established that the nature of the assessments administered impacts on the probability of students engaging in contract cheating, which entails devious acts such as paying others to do work on the student's behalf. Gandi *et al.* [19] gave lit of measures one needs to observe in order to minimize cases of plagiarism. This page may be useful for both teachers and learners to comprehend what kind of actions might be regarded as plagiarism and how they might be avoided. Devlin and Gray [15], the author presented current tools and technology used in the detection of plagiarized materials in natural as well as programming languages. These tools can be useful in combating academic dishonesty especially in so far as plagiarism is concerned. Sallis *et al.* [16] conducted a qualitative study to achieve the goal of the study that was to "find out why students plagiarize". They stress the necessity to tackle the causes of plagiarism, including misconceptions, lack of time, or comprehension difficulties. Gniazdowski *et al.* [20] elaborated he approach based on the utilization of machine learning for semantic plagiarism detection in VHDL code. Externally the code may appear very dissimilar, but by comparing the inner content it is possible to detect plagiarism when code has been changed a lot. A report on academic integrity by International Center for Academic Integrity was released in 2024 and sets out more on the revolutions around plagiarism.

Mirza *et al.* [21] explored case in digital design courses involving the plagiarism of VHDL code. The observation reveal the importance of proper methodology in plagiarism detection in this discipline. Many scholars have written about plagiarism especially [22], they elaborated on the problem, as well as the lapses that are evident in the instructions given and the resultant programming assignments that make them to result in plagiarism. Liu *et al.* [23] conducted a study in which they compared different source code plagiarism detection engines and further summarized in tabular form, the comparative results, which were most useful to understand the efficacy of the different methodologies which were used. Mccabe *et al.* [24] described GPLAG, a method of identifying the act of software plagiarism based on a program dependence graph analysis. This method is useful in comparing programs and as such is useful in detecting more complex forms of plagiarism, where the code may be similar, but have a different logic. Roy *et al.* [25] have tried to examine cheating rate in academic institutions over the last decade, thus helping to analyze the level and tendencies of academic dishonesty. Whale *et al.* [26] introduced NCAD, a demonstrative tool to effectively detect close I-clones by PPP and CN. This tool is especially useful for cases where the code has been written to be as hidden as possible from a virus scanner's perspective. Whale *et al.* [26] conducted on analysis of underlined frequency and trends of source code piracy by the undergraduate computer science students. From their study, they have proposed the need for increased awareness and implementation of academic misconduct in computer science departments. Last, Bretag *et al.* [27] addressed the issue of comparing programs to determine the existing similarity in large population, something that has a lot of connection with the detection of code plagiarism.

Several researchers have made significant contributions to the field of plagiarism detection in programming languages and technical writing. Here, we highlight some of the most impactful work Ming *et al.* [28] conducted a comprehensive review of plagiarism detection tools for both natural and programming languages. The authors analyzed various plagiarism detection techniques and tools and found that the structural metric techniques generally outperform attribute counting methods in identifying code similarities. Mccabe *et al.* [24] compared the effectiveness of attribute counting and structural metric approaches. The authors conducted comparative analysis of different plagiarism detection methodologies and they found the structural metrics are more robust in detecting complex forms of plagiarism, including code rearrangement and variable renaming. Whale *et al.* [26] addressed the challenge of identifying program similarity in large populations and he developed algorithms for large-scale code similarity detection. After analyzing the author proposed scalable methods that could handle substantial code repositories effectively. Novak *et al.* [6] performed a systematic review of source-code similarity detection tools used in academia. The authors conducted a comprehensive survey of existing plagiarism detection tools and their applications and they highlighted the need for more sophisticated methods to combat evolving plagiarism techniques. Jones and Williams [14] explored plagiarism detection in VHDL code and Investigated plagiarism detection methods specific to hardware description languages. They emphasized the unique challenges posed by the syntax and structure of hardware description languages in plagiarism detection.

3. METHOD

After the literature review is finished, one attribute counting methodology and one structural metric approach must be selected in accordance alongside the assignment's goals listed in the introduction of this work. This section looks closely at a number of the two types of methods first, then selects one and gives a reason for the choice along with some explanation for why the other possibilities were not chosen.

3.1. Attribute counting method

This was not an easy conclusion to make because there is a dearth of evidence regarding which attribute counting method actually works the best. However, McCabe's method was seen as a good and legitimate way to apply in the attempt to detect plagiarism because it is widely accepted as an industry standard for calculating code complexity. It is now "able to detect transposed subsequences" and is currently not plagued by the "difficulties faced by the well-known algorithms in detecting similarities in the presence of block-moves," according to Wise [11], who used the GST method in YAP3.

3.1.1. Maccabeus's cyclomatic algorithm

An internet search yielded results for a program named resource standard metrics (RSM). It makes it possible to compute maccabeus's cyclomatic complexity on a segment of Verilog source code. The cyclomatic challenges associated with each of the dozen strategies in the benchmark set were determined in order to evaluate the attribute counting approach. Next, the variations between each algorithm's complexity and threshold are documented using the previously determined thresholds. Figure 1 demonstrates the Maccabeus's cyclomatic complexity graph. The primary issue with this difficulty lies in how it just counts the amount of paths with linear independence that flow via the code rather than taking into account each token that contributes to the overall structure of an algorithm. This means that even when the code is known to be obviously different, there is a chance that an algorithm's complexity will fall below the threshold and lead to an incorrect classification.

3.2. Structure metric method

Verilog-Greedy-String-Tiling, or jGST, will be the new name for the created Verilog tool. First, the tool's needs are:

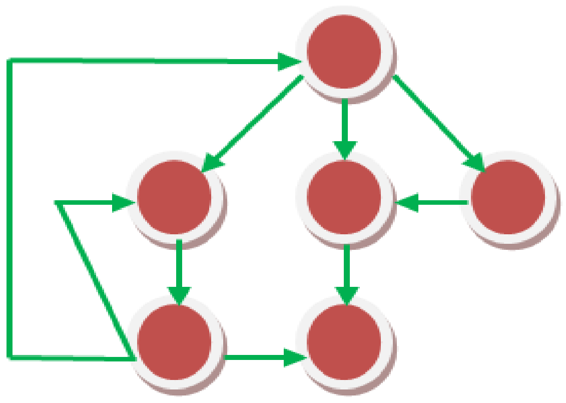

Figure 1. Maccabeus's cyclomatic complexity graph

To give the user the option to select the source and target Verilog code files-that is, the files that will be compared. To provide a minimum match length (MinML) option for the user. To produce a numerical plagiarism score comparing two source code fragments. To function on the PC operating system. With the exception of number seven, "changing the order of independent statements," Wise discovered that YAP could effectively handle each of the aforementioned issues. Each SIM program is first digested using the dictionary analyzer to generate a series of integers known as tokens. A dynamic programming string alignment technique is then used to compare the token sequences. Using this method, a score is initially assigned to each alignment of characters. A match, for instance, scores 1, and a mismatch, -1. The resemblance of texts produced in C, Java, Pascal, and natural language is then tested using the score between two sequences, which is defined as the maximum score among all alignments. According to Wise [11], YAP3 is now "able to detect transposed subsequences" and is not hampered by the "difficulties faced by the well-known algorithms in detecting similarities in the presence of block-moves" because it started using the GST technique. Overall, the generalized substitution approach that YAP3 utilizes was decided to be adopted because it is possible to create a program in this project that can catch piracy caused by every single one of the strategies stated above. It was acknowledged that employing this approach would make understanding the project's ultimate conclusions considerably more challenging, if not impossible. Because it does not take relocated blocks into account, Whale [8] explains that "the biggest similar subsequence is not a perfect way to determine the amount of consistency that exists between like [token] patterns." Put differently, Plague is unable to detect instances in which independent assertions are rearranged to conceal plagiarism. As it was just shown, the GST mechanism that YAP3 employs can handle this task. Four more "desirable" conditions were identified in addition to the ones mentioned above. These were: To show, upon their comparison, any substrings that match between the source and target files. The ability to store, in a ranking order, all of the plagiarism scores from comparisons performed in a single session.

The ability to reset the tool, eliminating the need to refresh the application in order to start a new session. Including the utility in a user-friendly graphical user interface (GUI). Token string conversion of both the original and target files has the first step throughout the process. Since this isn't a genuine component of the GST method, it was decided to use a pre-written tokenizer in order to save time. After that, one such instance was found5 and modified for this project's use. Additionally, it was discovered that this tokenizer could eliminate string constants and comments.

3.3. Method

The proposed system is designed to meticulously identify instances of plagiarism within both textual content and Verilog code by leveraging a range of linguistic characteristics extracted from tests through the utilization of the WordNet lexical database. The overarching goal of the system is to achieve the highest possible accuracy in discerning instances of plagiarism, and to this end, it employs distinct features that cover a spectrum of plagiarism manifestations. These features include but are not limited to modifications in text structure, word substitutions with their synonyms, and the simultaneous application of these techniques. The systematic framework of the proposed system is structured into three sequential stages, namely preprocessing, detailed analysis, and post-processing. Each of these stages plays a crucial role in enhancing the overall efficacy of the system in identifying and characterizing plagiarism. A pivotal component of the system is the creation of a database, which serves as the foundation for training deep learning models tailored specifically for the detection of text plagiarism. This database is meticulously curated, encompassing a comprehensive set of similarity features that span various lexical, syntactic, and semantic aspects of text. Each entry within this database represents a distinct similarity case, contributing to the nuanced learning of the deep learning models.

In the context of Verilog code, the system introduces the `GSTalgorithm` class, which is characterized by two key attributes. The first, `minMatchLength`, is self-explanatory and sets a threshold for the minimum length of matching sequences. The second attribute is a list of reserved words (`resWords`), which, excluding the names of built-in functions, constitutes the Verilog lexicon. Notably, these reserved words serve as the exclusive tokens considered during the construction of token sequences from both source and target files. This strategic approach aims to eliminate tokens that do not belong to the Verilog lexicon, thereby refining the analysis to focus on language-specific elements.Further granularity is introduced through the instantiation of the `GST token` class, representing individual tokens in the token sequences. Each instance encapsulates the token's name and a binary indicator of whether it is marked. As shown in Figure 2 illustrates the flow chart of parsing steps in plagirisim. The subsequent instantiation of the `GSTtile` class facilitates the representation of sequences of matched tokens between source and target files. Each instance of `GSTtile` is constructed with three attributes: `sourceIndex` and `targetIndex`, indicating where the matched sequence begins in each file, and a `length` value denoting the extent of the matched sequence.

In summary, the proposed system adopts a comprehensive and nuanced approach, incorporating linguistic features, deep learning models, and targeted processing stages to accurately detect and characterize plagiarism in both textual content and Verilog code.

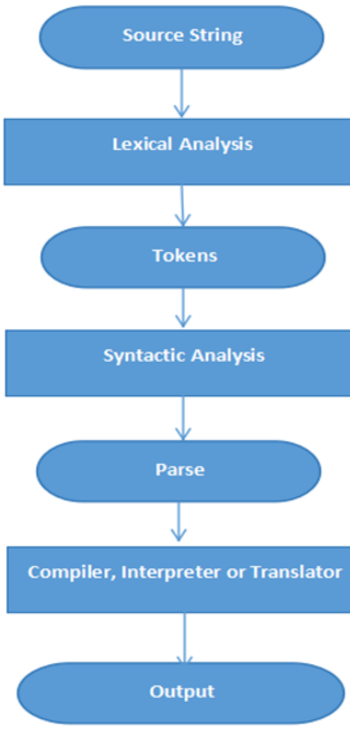

Figure 2. Parsing steps flowchart

3.4. Collecting datasets

The creation of a data collection is now necessary after the chosen characteristic counting and structural evaluation methods are implemented. The finished set will have two uses. It will first give Verilog algorithms for determining an acceptable limit for each technique, and then it will contain strategies for creating an experiment set. It was determined to take into consideration six fundamental algorithmic concerns and construct an information set from representations of those sets in order to reflect a concept of recognizing plagiarism in programs that might have been turned in within the context of an academic task. The following defines the six levels of the aforementioned spectrum:

- L0: Nothing has changed; this is the initial software.
- L1: Shows the adjustments made to the indentation and remarks.
- L2: Indicates how the IDs and level 1 have changed.
- L3: Indicates modifications to level 2 and declarations (such as adding new constants, rearranging functions, and altering the locations of declared variables, among other things).
- L4: Depicts the modifications made to program modules and level 3 (e.g., combining two operations into one or adding fresh functions).
- L5: Indicates what adjustments were made to stage 4 and the program expressions (e.g., using FOR rather than WHILE).
- L6: Depicts the adjustments made in stage 5 and the selection logic (i.e., expression alterations).

3.5. Testing and analysis

In the upcoming testing and analysis chapter, a comprehensive evaluation of the structure metric method will be undertaken, necessitating the determination of four distinct minimum match lengths (MinML) for thorough testing (refer to page 9 for details). This entails the establishment of four separate thresholds, each corresponding to a specific minimum match length. To identify the threshold for the structure metric method, the highest modified level within each of the six sets of plagiarized algorithms is intentionally withheld, mirroring the approach employed in the threshold identification process of the attribute counting method. These reserved values, representing the peak modified level for each algorithm set, serve as essential positive test data in subsequent analyses. Figure 3 show the spectrum of six level plagiarism steps. The resulting data is then visually presented through six graphs, with each data point categorized into one of two sets based on whether it pertains to an algorithm pair known to involve plagiarism. Subsequently, quartile ranges are computed for each set. The actual threshold is determined as the average value between the upper quartile of the non-plagiarized set and the lower quartile of the plagiarized set. This methodology is elucidated in the accompanying box plot diagram. In practical terms, if a data value surpasses the calculated threshold, it is deemed to be indicative of plagiarism. Conversely, if the value falls below or on the threshold, it is classified as non-plagiarized. The utilization of quartile ranges in threshold calculation offers a distinct advantage by discounting anomalies within each set, aiming to yield a more precise and reliable threshold value. This approach enhances the robustness of the structure metric method's ability to discern instances of plagiarism accurately.

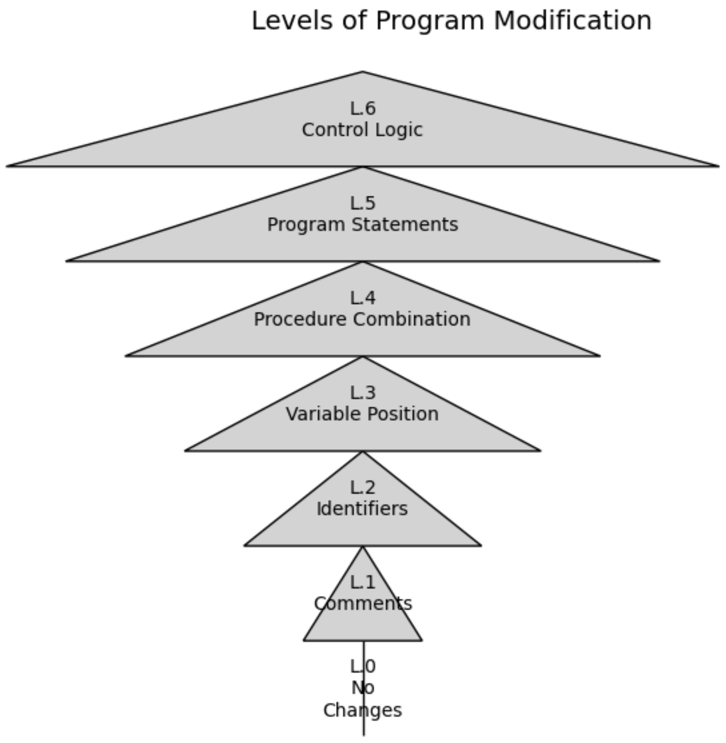

Figure 3. Six-level plagiarism spectrum

4. RESULTS AND DISCUSSION

In the envisioned system aimed at detecting plagiarism in both textual content and Verilog code, diverse linguistic characteristics are harnessed by leveraging the WordNet lexical database. The primary objective of the system is to achieve the utmost accuracy in identifying instances of plagiarism within text and Verilog code. Figure 4 demonstrates the detection accuracy across various datasets, illustrating the system's performance in identifying different forms of plagiarism, including structural alterations and synonym substitutions. These distinctive characteristics encompass various manifestations of plagiarism, including alterations to text structure, word substitution with synonyms, and the simultaneous utilization of both strategies. The system's workflow is primarily categorized into three stages: preprocessing, in-depth analysis, and post-processing.

The effectiveness of the proposed system hinges on the establishment of a database, integral for training deep learning models tailored to detect text plagiarism. This database is meticulously crafted, taking into account a myriad of similarity features that capture diverse lexical, syntactic, and semantic aspects of text. Each entry in this database encapsulates a unique similarity case, contributing to the comprehensive learning of the deep learning models. In summary, the proposed system's architecture is intricately designed to uncover instances of plagiarism in both textual and Verilog code by extracting nuanced linguistic characteristics from texts through the WordNet lexical database. The systematic process involves preprocessing, detailed analysis, and post-processing, supported by a database enriched with diverse similarity features for effective deep learning model training.

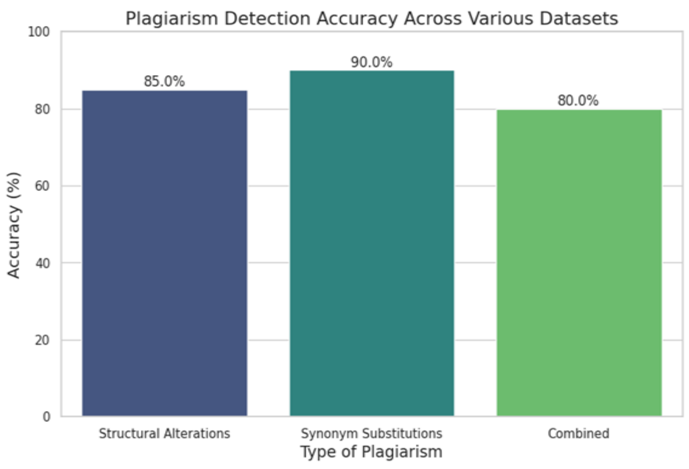

Figure 4. Detection accuracy across various datasets

Each entry in this database represents a distinctive similarity case, ensuring a comprehensive approach to plagiarism detection. These results prompt several important considerations. First, the ability of the system to effectively identify structural changes suggests that our preprocessing techniques, which normalize and tokenize the text, are robust. This is crucial because many traditional plagiarism detection systems struggle with paraphrased content.

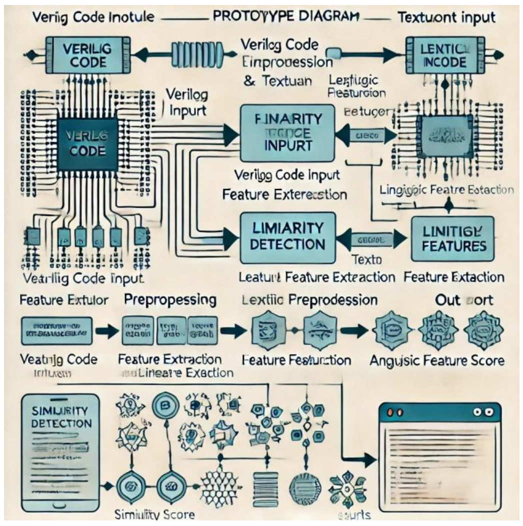

Figure 5. A prototype model for plag detection

Additionally, the findings emphasize the value of the comprehensive database we established. By including diverse similarity features that encapsulate lexical, syntactic, and semantic elements, we can better train our deep learning models. Each entry in our database provides a unique perspective on similarity, reinforcing the system's learning capabilities. The implications of these findings are significant for both academic integrity and software development. By improving plagiarism detection in Verilog code, we address a gap in current literature, particularly for fields heavily reliant on code sharing and collaboration. This system not only aids educators and researchers in maintaining ethical standards but also fosters a culture of originality in technical disciplines.

5. CONCLUSION AND FUTURE WORK

Our research underscores the necessity for sophisticated plagiarism detection mechanisms that can adapt to various forms of content manipulation. The integration of WordNet and the systematic approach to feature extraction position our system as a leading tool in the ongoing fight against plagiarism, deserving of further exploration and potential implementation in educational settings. Ultimately, the similarity of the file pairings is expressed as a single correlation score. A higher score indicates greater similarity and potential for plagiarism between the files in a pair. By enabling the expert to focus on dozens of lines of code in dozens of files instead of hundreds of thousands of lines of code in hundreds of files, CodeMatch lessens the work required. It has been observed that the tool does not display any unnecessary content for the user, such as logos, slogans, or unsolicited information. Additionally, jGST's arrangement makes sense; on-screen information is arranged from upper left to lower right of the window. While some terminology is provided within the system, most of the concepts are thought to be highly understandable and self-explanatory. Take the menu option select source file, for instance. If the ClearResults option has been selected, for example, and the findings on screen have begun to be deleted the utility does not warn the user. This would be made mandatory if this tool was to be upgraded. No further assistance or documentation has been provided because it is believed probable users are aware with the broad concepts underlying the GST procedure and that the tool is quite easy to use. Once more, this choice would have to be reevaluated if the tool were to be enhanced. It is difficult to propose any potential future projects that could arise after this one is finished. If attribute counting techniques are to be employed once more in a study of a comparable nature, a more comprehensive analysis and assessment are absolutely necessary. It is believed that trying to identify stronger, more tangible results using both types of methods isn't always the best course of action. It became evident that the project's ability to show that the strategy is ineffective was its main advantage in doing this. Of course, this is still a legitimate outcome, even though it might not be exactly what was intended. Moreover, future studies could explore the integration of additional linguistic resources or machine learning techniques to further enhance the system's accuracy. Additionally, evaluating the system's performance in real-world scenarios would provide valuable insights into its practical applicability.

ACKNOWLEDGMENTS

I thank my friends family members and specifically my honorable dean Dr. S.P. Chokkalingam and our Registrar Dr. E. Kannan for his valuable support.

FUNDING INFORMATION

No funding involved

AUTHOR CONTRIBUTIONS STATEMENT

This journal uses the Contributor Roles Taxonomy (CRediT) to recognize individual author contributions, reduce authorship disputes, and facilitate collaboration.

Name of Author	C	M	So	Va	Fo	I	R	D	O	E	Vi	Su	P	Fu
Dr. Kiran Kumar Manivannan	✓		✓		✓	✓		✓		✓			✓	✓
Dr. Nalayini C. M.						✓		✓	✓		✓			
Dr. Sathya V.		✓	✓				✓			✓	✓		✓	
Dr. Kumar G.				✓	✓				✓	✓				
Dr. Dinesh Babu M.					✓		✓			✓		✓		

C	:	**C**onceptualization	I	:	**I**nvestigation	Vi	:	**Vi**sualization
M	:	**M**ethodology	R	:	**R**esources	Su	:	**Su**pervision
So	:	**So**ftware	D	:	**D**ata Curation	P	:	**P**roject administration
Va	:	**Va**lidation	O	:	Writing - **O**riginal Draft	Fu	:	**Fu**nding acquisition
Fo	:	**Fo**rmal analysis	E	:	Writing - Review & **E**diting			

CONFLICT OF INTEREST STATEMENT

Authors state that there is no conflict of interest.

INFORMED CONSENT

We have obtained informed consent from all individuals included in this study.

DATA AVAILABILITY

The data that support the findings of this study are openly available in Kaggle data set.

REFERENCES

[1] I. Smit, E. Naudé and B. Zulu, "A detection process to create awareness of source-code plagiarism among students using it to pass introductory programming," *The Independent Journal of Teaching and Learning*, vol. 19, no. 1, pp. 79-92, 2024, doi: 10.17159/ijtl.v19i1.18854.

[2] R. S. Mehsen, M. M. Kazi and H. Joshi, "Detecting source code plagiarism in student assignment submissions using clustering techniques," *Journal of Techniques*, vol. 6, no. 2, 2024, doi: 10.51173/jt.v6i2.1851.

[3] A. A. Pandit and G. Toksha, "Review of plagiarism detection technique in source code," *International Conference on Intelligent Computing and Smart Communication 2019*, 2020, pp. 393-405, doi: 10.1007/978-981-15-0633-8_38.

[4] J. Yasaswi, S. Kailash, A. Chilupuri, S. Purini and C. V. Jawahar, "Unsupervised learning based approach for plagiarism detection in programming assignments," *Proceedings of the 10th Innovations in Software Engineering Conference*, 2017, pp. 117-121, doi: 10.1145/3021460.3021473.

[5] R. S. Mehsen, M. M. Kazi and H. Joshi, "Detecting source code plagiarism in student assignment submissions using clustering techniques," *Journal of Techniques*, vol. 6, no. 2, 2024, doi: 10.51173/3/jt.

[6] M. Novak, M. Joy and D. Kermek, "Source-code similarity detection and detection tools used in academia: a systematic review," *Acm Transactions on Computing Education*, vol. 19, no. 3, 2019, doi: 10.1145/3313290.

[7] K. P. Gomes, S. Nasser Matos and Tarcizio Alexandre Bini, "BIOPLAG: An approach to detect programming plagiarism," Anais da Academia Brasileira de Ciências," vol. 95, no. 3, 2019, doi: 10.1590/0001-3765202320220684.

[8] M. Kaya and S. Ayşe Özel, " Integrating an online compiler and a plagiarism detection tool into the moodle distance education system for easy assessment of programming assignments," *Computer Applications in Engineering Education*, vol. 23, no. 3, pp. 363-373, 2014, doi: 10.1002/cae.21606.

[9] H. Cheers, Y.g Lin and W. Yan, "Identifying plagiarised programming assignments with detection tool consensus," *Informatics in Education*, vol. 22, no. 1, 2023, doi: 10.15388/infedu.2023.05.

[10] M. Hoq, Y. Shi, J. Leinonen, D. Babalola, C. Lynch, T. Price and B. Akram, "Detecting ChatGPT-generated code submissions in a CS1 course using machine learning models," *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V.1*, 2024, pp. 526-532, doi: 10.1145/3626252.3630826.

[11] T. Ahmed, "Exploring mathematical models and algorithms for plagiarism detection in text documents: a proof of concept," *Research Square*, 2014, doi: 10.21203/rs.3.rs-3973392/v1.

[12] Z. I. Azhari, S. Setumin, E. Noorsal and M. H. Abdullah, "Digital image enhancement by brightness and contrast manipulation using Verilog hardware description language," *International Journal of Electrical and Computer Engineering (IJECE)*, vol. 13, no. 2, pp. 1346-1357, doi: 10.11591/ijece.v13i2.pp1346-1357.

[13] V. A. Romanovych *et al*., "basic principles of prevention and detection of academic plagiarism in a medical university (higher medical educational institution)," *ВісникНаукиТаОсвіти*, 2024, doi: 10.52058/2786-6165-2024-4(22)-724-735.

[14] U. Bandara and G. Wijayrathna, "Detection of Source Code Plagiarism Using Machine Learning Approach," *International Journal of Computer Theory and Engineering*, vol. 4, no. 5, 2021.

[15] M. Devlin and K. Gray, "In their own words: a qualitative study of the reasons Australian university students plagiarize," *Higher Education Research & Development*, vol. 26, no. 2, pp. 181-198, 2007, doi: 10.1080/07294360701310805.

[16] P. Sallis, A. Aakjaer and S. MacDonell, "Software forensics: old methods for a new science," *Proceedings 1996 International Conference Software Engineering: Education and Practice*, Dunedin, New Zealand, 1996, pp. 481-485, doi: 10.1109/SEEP.1996.534037.

[17] S. Mahmud, T. Bretag and T. Foltýnek, "Students' Perceptions of Plagiarism Policy in Higher Education: a Comparison of the United Kingdom, Czechia, Poland and Romania," *Journal of Academic Ethics*, vol. 17, pp. 271-289, 2019, doi: 10.1007/s10805-018-9319-0.

[18] C. Park, "In Other (People's) Words: Plagiarism by university students--literature and lessons," *Assessment & Evaluation in Higher Education*, vol. 28, no. 5, 2003, doi: 10.1080/02602930301677.

[19] N. Gandhi, G. Kaushik Gopalan and P. Prasad, " A support vector machine based approach for plagiarism detection in python code submissions in undergraduate settings," *Frontiers in Computer Science*, 2024, doi: 10.3389/fcomp.2024.1393723/full.

[20] Z. Gniazdowski and M. Boniecki, "Detection of a Source Code Plagiarism in a Student Programming Competition," *Zeszyty Naukowe WWSI*, vol. 13, no. 21, pp. 74-94, 2018, doi: 10.48550/arXiv.1912.08138.

[21] O. M. Mirza, M. Joy and G. Cosma, "Style analysis for source code plagiarism detection — an analysis of a dataset of student coursework," *2017 IEEE 17th International Conference on Advanced Learning Technologies (ICALT)*, Timisoara, Romania, 2017, pp. 296-297, doi: 10.1109/ICALT.2017.117.

[22] T. Lancaster and F. Culwin, "A Comparison of source code plagiarism detection engines," *Computer Science Education*, vol. 14, no. 2, pp. 101-112, 2010, doi: 10.1080/08993400412331363843.

[23] C. Liu, C. Chen, J. Han and P. S. Yu, "GPLAG: detection of software plagiarism by program dependence graph analysis," *Proceedings of the 12th AcmSigkdd International Conference on Knowledge Discovery and Data Mining*, 2006, pp. 872-881, doi: 10.1145/1150402.1150522.

[24] D. L. Mccabe, L. K. Trevino and K. Butterfield, "Cheating in academic institutions: a decade of research," *Ethics & Behavior*, vol. 11, no. 3, pp. 219-232, 2010, doi: 10.1207/S15327019EB1103_2.

[25] C. K. Roy and J. R. Cordy, "NICAD: Accurate detection of near-miss intentional clones using flexible pretty-printing and code normalization," *2008 16th IEEE International Conference on Program Comprehension*, Amsterdam, Netherlands, 2008, pp. 172-181, doi: 10.1109/ICPC.2008.41.

[26] G. Whale, "Identification of program similarity in large populations," *The Computer Journal*, vol. 49, no. 4, pp. 411-422, 1990, doi: 10.1093/comjnl/33.2.140**.**

[27] T. Bretag *et al*., "Contract cheating and assessment design: exploring the relationship," *Assessment & Evaluation in Higher Education*, vol. 44, no. 5, 2019, doi: 10.1080/02602938.2018.1527892.

[28] J. Ming, F. Zhang, D. Wu, P. Liu and S. Zhu, "Deviation-based obfuscation-resilient program equivalence checking with application to software plagiarism detection," in *IEEE Transactions on Reliability*, vol. 65, no. 4, pp. 1647-1664, Dec. 2016, doi: 10.1109/TR.2016.2570554.

BIOGRAPHIES OF AUTHORS

Dr. Kirankumar Manivannan is currently working as assistant professor (senior grade) in the school of electronics engineering at Vellore Institute of Technology, Chennai. He has completed his doctorate in Anna University. His main research interests are VLSI design, sensors, wireless communication. He has published many papers in international and national journals. He is a reviewer of some International Conferences. He can be contacted at email: kiransun5@gmail.com.

Dr. Nalayini C. M. is currently working as an assistant professor in the department of information technology at Velammal Engineering College. Her teaching experience is 19 years in total. She has completed her Ph.D. degree in information and communication engineering at Anna University, Chennai, India. Her current research interest includes network security, machine learning, deep learning and blockchain technologies. She has published various text book and book chapters. She has published many research articles in international journals and conferences. She has won best paper award for the blockchain research paper. She has received best researcher award by ISSN awards for the article titled "A Novel Dual Optimized IDS to detect DDoS attack in SDN using Hyper Tuned RFE and Deep Grid Network". She is one of the recognized reviewers for many reputed journals and book chapters. She is one of the faculty ambassadors for the Hyperledger India Chapter Student Society. She can be contacted at email: nalayini@velammal.edu.in.

Dr. Sathya V. is an engineer by qualification, having completed her doctorate in the area of wireless sensor networks from Anna University, Chennai and masters in computer science and engineering from PRIST University. She has received her bachelor's in computer science and engineering from Anna University. She is presently working as assistant professor (Sr.Grade) in the department of CSE at Vel Tech Rangarajan Dr. Sagunthala R&D Institute of Science and Technology, Chennai. Her areas of research interest include sensor networks, malware detection, sandboxing technology, artificial intelligence, data science, machine learning and deep learning. She has published 50+ articles in reputable journals. She is a member of IEEE and ISTE. She can be contacted at email: saro.sath@gmail.com.

Dr. Kumar G. holds a B.E. in electronics and communication (SRM Engineering College, 2001), an MBA (Bharathidhasan University, 2007), and a Ph.D. in management (SRM Institute of Science and Technology, 2009). His doctoral research focused on mobile banking adoption in Chennai. With 14+ years of teaching experience at SRM's Faculty of Management, he has published 70+ articles in reputable journals. Additionally, he served as a programmer (2003-2009) with system administration and documentation responsibilities. His areas of expertise include business analysis, analytics, programming (Python, R), data structures, storytelling, statistics, big data, SQL, and machine learning. He has held the role of domain coordinator-systems (2022-2023) and currently mentors and leads classes for management students. He can be contacted at email: kumarg@srmist.edu.in.

Dr. Dinesh Babu M. received his B.E. (mechanical engineering) from the University of Madras in 2002, and M. Tech. (energy systems engineering) from V.I.T. Vellore in 2004. He obtained his Ph.D. from the college of engineering, Guindy Campus, Anna University, Chennai -25 for his research in energy engineering. He started his teaching career as a lecturer in 2004 and has 20 years of experience in both teaching and research. Currently, he works as a professor at the Rajalakshmi Institute of Technology, Chennai. He has published in 64 International Journals. He has guided over 30 UG students and guided one Ph.D. research scholar. His areas of interest are solar energy, alternative fuels, and CFD. He can be contacted at email: dinesh198014@yahoo.com.