Detecting Plagiarism in Student Assignments using Source Code Analysis

SERHII MYRONENKO, YELYZAVETA MYRONENKO
System Design Department, Institute for Applied System Analysis,
National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute",
37, Peremohy Ave, Kyiv, 03056,
UKRAINE

Abstract: - The issue of accurately detecting semantically equivalent code remains a current problem for students, teachers, and researchers. The goal of this article was to develop an alternative method for checking software code for plagiarism using abstract syntax trees. To achieve this, an analysis of existing modern methods of automatic comparison of original and modified program code for plagiarism detection was conducted, focusing on abstract syntax trees, approaches based on machine learning, and token-based analysis. Systems such as MOSS, JPlag, Codequiry, and Plaggie were reviewed. As a result of this work, it can be noted that the proposed methods have sufficiently high indicators in the range of 79-85% for semantically equivalent code and 93-95% similarity in cases demonstrating hash violation problems, which shows their potential effectiveness in solving various tasks related to the detection of software code plagiarism. The practical value of the research lies in the possibility of using the proposed method for checking small student assignments for originality. Future research on this topic could include adding functionality for document collection, reducing the time required for document verification, expanding the number of programming languages, and improving system efficiency while operating large code fragments.

Key-Words: - Source Code Plagiarism, BERT, Code2Vec, PLP, Transformers, AST.

Received: January 19, 2024. Revised: July 2, 2024. Accepted: August 1, 2024. Published: September 4, 2024.

1 Introduction

Currently, the problem of plagiarism in student's works at higher educational institutions is quite serious. This is linked to the fact that, as indicated in articles, [1], [2], [3], the majority of experienced students do not consider "borrowing" someone else's work without citation as a violation of academic integrity. Furthermore, some students rely on the notion that information should be free and can be "reused." As a result, learning, which is already advantaged by a large number of practical and laboratory works, becomes significantly time-saving, even if it involves copying articles from the Internet, which only takes a few minutes compared to independent problem analysis, [4]. Another argument for indulging in plagiarism for a student is the awareness that their peers are also copying works and receiving high grades for it.

To effectively assess the scale of the problem of appearances in student works, existing research on this topic was analyzed, which fully demonstrates the problem's relevance. Several sociological experiments, which were conducted both for students and for lecturers at higher educational institutions, were analyzed. Also were analyzed articles, [2], [3], [5], [6], [7], which are dedicated to the study of student psychology and their attitude towards the problem of plagiarism. Several key thoughts of the authors, which underline the seriousness of the problem of plagiarism and, as a consequence, the relevance of this work, will be presented.

In the work, [5], the author considers that plagiarism and cheating are natural, and people are not particularly distinguished in using such a deception tactic to gain a competitive advantage. At the same time, the author points out that "reasonable intervention" in this process is possible, and numerous resources for lecturers can help create a culture of integrity. The authors, [2], present similar thoughts, and list several reasoned causes that lead to the violation of academic integrity rules:

- Lack of time;
- Helping another in completing a task;
- The student does not understand the scope of work required and is overwhelmed with the task;
- External pressure (for example, parents pushing for high grades);

- The student is unable to complete the task independently;
- Laziness;
- The belief is that all other students are also cheating.

The authors emphasize that the problem of plagiarism needs to be solved at several levels – from plagiarism detection systems to appropriate normative acts and official rules and procedures. It is also worth paying attention to the article, [2], which describes in detail the students' point of view on the problem of plagiarism. The authors of this work fully share the higher education thoughts and assertions and also present several key aspects:

- Students do not always clearly understand what constitutes plagiarism. A comparable observation was also noted in the article, [8];
- In different countries, there are different understandings of plagiarism (Thus, without clear rules, a student may not fully understand what is considered plagiarism in one educational institution).
- In this study, a sociological investigation was conducted to understand students' perceptions of what constitutes plagiarism in programming code. The findings revealed that a significant number of students do not fully grasp the concept of plagiarism. For instance, according to materials from the article, more than 50% of experienced students believe that taking program code from a book and rewriting it in another language is not plagiarism. However, 94% of respondents acknowledge that directly copying program code from a book violates academic integrity rules.

After analyzing various articles, [2], [3], [5], [6], [7], the following conclusions can be drawn (all statistical data are taken from the materials of the works):

- A large number of students significantly underestimate the number of plagiarism incidents in other works, which encourages them to violate academic integrity rules;
- The majority of experienced students (more than 85%) are neutral or positive towards the practice of copying other works;
- Nearly 70% (mostly first-year students) support the absence of punishment for plagiarism or limited verbal warnings;

According to information from the article, [9], most educators, and specifically 97% of experienced ones, do not react to instances of academic dishonesty in practice, with 60% of participants limiting their response to verbal warnings. Moreover, only 5% of experienced participants allow the use of additional printed materials during exams and other types of assessment works, confirming the effectiveness of this practice in combating plagiarism.

To highlight the prevalence of code plagiarism in recent years, the following statistics from researchers are provided:

- From the information from the article, [10], it was confirmed that whereas in the 1940s only 20% of college students admitted to cheating, nowadays the percentage has increased to 75 - 98%.
- Approximately 42% of computer science students admitted to copying code from other students – that is the result of the survey, [11].
- Moreover, according to the article, [12], it was proven that from 50% to 79% of students will plagiarize at least once in their academic career.

As a premise, it can be concluded that the issue of plagiarism in student works requires detailed attention. The main justifications for performing this research are maintaining academic integrity and the potential support of academic institutions.

In the field of Information Technology (IT), plagiarism can be broadly categorized into two groups:

- Text plagiarism;
- Source code plagiarism.

A more detailed classification of plagiarism types is presented in Figure 1, based on the materials of works, [13], [14].

Notably, this work will focus on one of these plagiarism groups, specifically source code plagiarism. Further, more detailed descriptions of the subgroups of program code plagiarism will be provided. These include:

- Refactoring of structure: The developer changes the order of functions and/or modifies the syntax without altering the program's source.
- Addition of additional tabulation, spaces, and comments, and further development of the work as their own.
- Change of programming language: The programmer rewrites the code in another programming language without acknowledging the authors of the original program code.

To prepare for this study, several works of other researchers on the topic of code plagiarism detection were analyzed. After careful studying of the selected works, it is worth mentioning some of them, particularly, [15], [16], [17] in more detail. The first work for analysis was an article, [15], about the DeepSim system. After analysis of this work, it can be stated that the main advantages of this system are feature learning, scalability, and generalization.

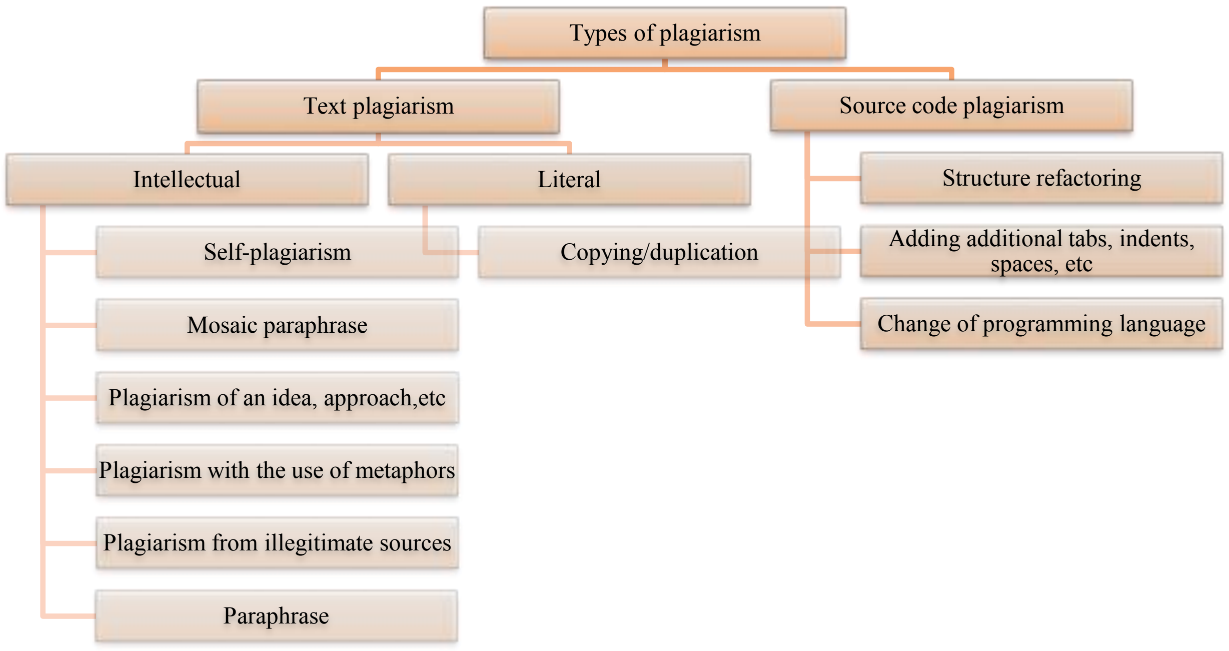
Types of plagiarism
Text plagiarism
Source code plagiarism
Intellectual
Literal
Self-plagiarism
Copying/duplication
Mosaic paraphrase
Plagiarism of an idea, approach,etc
Plagiarism with the use of metaphors
Plagiarism from illegitimate sources
Paraphrase
Structure refactoring
Adding additional tabs, indents, spaces, etc
Change of programming language

Fig. 1: Classification of plagiarism types

On the other hand, the main disadvantages of this system are overfitting, interpretability, and robustness.

In the article, [16], the use of natural language processing methods was explored. The main pros of this work are code reusability, semantic comprehension, and context relevance. At the same time, the cons of this research are domain specificity, limited vocabulary, and semantic gap.

It is also worth noting research, [17]. The authors of this article studied the use of different embedding techniques to represent code snippets in a continuous vector space for efficient code plagiarism detection. The main concerns for this study are data quality, hyperparameter tuning, and interpretability, whereas, on the other hand, the potential benefits of this work are semantic understanding, efficiency, scalability, and robustness.

The objective of this research was to propose an alternative method for verifying program code for plagiarism using abstract syntax trees and neural networks. Consequently, the following tasks of this research were identified:

- To analyze popular systems for analyzing program code for plagiarism and identify the most effective among them;
- To determine the most effective system in terms of versatility (if possible);
- To propose an alternative approach for solving the task of detecting plagiarism in program code using abstract syntax trees and neural networks;
- To compare the results obtained from the analysis.

2 Methodology

This research is based on data collected from existing systems and methods for detecting similarities in program code, necessary for analyzing their effectiveness in identifying plagiarism in program code. The work employs and combines a range of theoretical research methods: logical method, analysis, synthesis, classification, generalization and analogy, comparison, induction, and interpretation. The methodological basis of the research consists of modern methodologies aimed at model structures designed for detecting plagiarism in program code.

In general, the procedure of this research can be divided into the following main stages:

1. Formation of a selection of existing systems for comparison;
2. Conducting a general comparative analysis of the possibilities of the systems;
3. Analysis of the results obtained with the subsequent selection of the two best systems;
4. Conducting a comparative analysis of the selected systems for accuracy in detecting program code plagiarism;

5. Analysis of the results of the comparison, identifying potential shortcomings of the systems;
6. Proposing alternative methods for detecting program code plagiarism and their practical comparison;
7. Analysis of the obtained results and conclusions.

As can be seen from Table 1, for comparison, we selected 4 existing systems currently designated for checking source code for plagiarism: MOSS, JPlag, Codequiry, and Plaggie.

The choice of MOSS, JPlag, Codequiry, and Plaggie for comparative analysis with other plagiarism detection systems can be justified based on several key factors that make these tools particularly relevant and representative in the context of code plagiarism detection:

2.1 Various Methodologies

Each of these systems employs different algorithms and methods for detecting similarities in code. This diversity allows for a comprehensive understanding of various approaches to detecting plagiarism in programming. To highlight the contrast between the selected systems, it is worth mentioning that MOSS uses a hash-comparison algorithm for detecting matches, whereas JPlag is based on searching for structural similarity, [18]. Whilst the Codequiry system browses a large library of samples, which includes web resources as well, Plaggie proposes customizable algorithms for code similarity detection, [19].

2.2 Popularity and Usage

The plagiarism checker software used for this study is reputable and applied in different educational institutions. For example, MOSS is becoming a benchmark in universities because of its reliability, which is a crucial point in any kind of study.

2.3 Support for Various Languages

The selected systems can be called versatile and applicable to a large variety of academic tasks due to their multiple programming language support. For example, JPlag is a solid choice for programming courses with a free selection of coding language, because the system is known for its “language independence”, [18].

2.4 Accessibility and Transparency

Transparency is one of the most important criteria for academic studies, due to the importance of detection procedure and its value to the whole learning process.

Moreover, the free accessibility of plagiarism detection tools which provide transparent results, makes them effective options for practical research and improvement of the educational process.

2.5 Different Target Audiences and Usage Scenarios

All the above mentioned tools are eligible for particular tasks and work environments.

To elaborate on the previous point, the Codequiry system can effectively evaluate modern programming assignments due to its focus on deep analysis and document comparison to the vast library of sources, [20].

Clear knowledge of each system functioning in different cases can significantly help educators and developers choose a plagiarism detection system according to their needs.

To summarize, the comparison of MOSS, JPlag, Codequiry, and Plaggie is based on the next important characteristics:

- Multiple programming languages support;
- Transparency;
- Ability to adapt to different use-case scenarios;
- Prevalence of use;
- variety of methodologies.

This comparison study will give educators and developers a comprehension of plagiarism detection tools as well as an understanding on methods of improving the educational process.

3 Results

As a part of this study, a comparative analysis of source code plagiarism systems was held based on selected criteria that can help evaluate the efficiency of the selected system. The study focuses on the following characteristics: ease of use, template code exclusion, scalability, clear and understandable results, capability to be used as a web service, support for multiple programming languages, and open source code availability. Definitions of the selected criteria are provided below.

Ease of Use

One of the most important criteria for plagiarism detection systems. Consequentially, systems with user-friendly and intuitive interfaces will be the obvious choice for teachers and students. As a result, ease of use greatly contributes to the increasing efficiency of the learning process and time-saving.

Template Code Exclusion

This function helps to ensure the accuracy of the code check and significantly reduces the number of false positives. Template code removal is especially useful for student assignment control as such tasks often include some generic code, [21].

Scalability

In our digital era, working with the large volumes of data is not a fancy feature, it is a necessity. The ability to work with large sets of documents simultaneously, fast adapting to different data sizes, and ensuring that the required computational power is provided are the necessary characteristics of an effective plagiarism detection system.

Clear and Understandable Results

Clear and understandable results are among the most vital sources of information for educators. Well-structured results provide the necessary feedback for teachers, which greatly helps to improve the educational process.

Capability to be used as a web service

A plagiarism detection system, that can be used as a web service provides great opportunities for people who work in different places. For example, such a system is a convenient choice for teachers who are regularly checking student assignments at home. Usually, web services are always highly maintained to stay up-to-date.

Support for Multiple Programming Languages

Support of multiple programming languages makes the plagiarism detection system more versatile and, as a result, it can be applied to a rather wide variety of educational tasks. Furthermore, it allows educational establishments to use a single plagiarism detection system for their work.

Open Source Code Availability in Plagiarism Detection Systems

Plagiarism detection systems with the open source code are a proper choice for educational institutions that value potential adaptability and transparency. Furthermore, the availability of the system's source code can help to evaluate its effectiveness, along with motivating the community to improve existing functionality and deliver new features.

Table 1 demonstrates the results of the comparative study.

To sum it up, after examining the obtained results, it can be ascertained that, according to the selected characteristics, the MOSS system is the most efficient.

Unfortunately, one of the candidates for evaluation, the Codequiry tool, could not be fully tested due to the absence of certain system functions in open access.

The next step in the comparative analysis is to compare the two best systems based on the results of the previous evaluation (MOSS and JPlag) for accuracy in detecting plagiarism (including all types of code similarity). The results of this comparison are presented in Table 2.

Table 1. Results of the Comparative Analysis of selected Plagiarism Detection Systems (MOSS, JPlag, Codequiry, Plaggie) according to the chosen criteria

№	Characteristics of the comparison	MOSS	JPlag	Codequiry	Plaggie
1	Ease of use	Yes	Yes	Yes (free functional was checked)	No
2	Template code exclusion	Yes	Yes	Yes	Yes
3	Scalability	No	No	No	No
4	Clear and understandable results; result demonstration	Yes, meets all requirements	Yes, meets all requirements	Yes (free functional was checked)	Yes, meets all requirements
5	Capability to be used as a web service	Yes	Yes	No	Yes
6	Open source code availability	Absent, but several projects are using MOSS algorithm	Absent	Absent	Absent
7	Support for multiple programming languages	23	6	More than 20	1

Table 2. Results of the Comparative Analysis of the Two Best Systems (MOSS and JPlag) for Accuracy in Detecting Plagiarism

Type of source code similarity	**MOSS**	**JPlag**
Identical code fragments	100 %	100 %
Changed variable names and/or their types	100 %	100 %
Added or removed individual operators	**92 %**	90 %
Semantically equivalent code	**64 %**	55 %

From the comparative table, it is evident that both systems effectively handle the detection of the 1st and 2nd types of code similarity. The 3rd type of code similarity was also detected quite effectively, but the MOSS system showed more precise results. Regarding the 4th type of similarity, the detection of semantically equivalent code, both systems show relatively low accuracy, but MOSS performs slightly better than JPlag.

At first glance, the MOSS system appears quite effective in detecting various types of plagiarism, but it is not as effective after considering the general principles of MOSS's operation and the creation of systems that can "deceive" MOSS by using critical system vulnerabilities, such as hash collision. The problem of hash collision lies in the fact that if a token is added to a hash window of length n, it significantly reduces the effectiveness of MOSS. In general, Mossad can generate up to 12 source code variants, and after checking for plagiarism, the original and MOSS do not show more than 25% similarity. Due to this vulnerability, this work proposes variants of program code testing for the presence of obfuscation, which would be robust against this vulnerability.

The approach proposed in this work involves a shift from the traditional MOSS algorithm to more o. The proposed block diagram is shown in Figure 2.

For implementing the algorithm depicted in Figure 2, two variants of realization for checking the presence of obfuscation are proposed. A multilayer perceptron was chosen for the first implementation that compares vector representations of programming code received from Code2Vec, [17], [22]. As the second option, the existing TreeBERT system, [23], [24], was modified to detect plagiarism in source code. TreeBERT system uses an enhanced vector representation, [24], [25], [26], [27], whereas the first realization is working with 'classic' implementation using standard representations received from Code2Vec.Each of the selected approaches has its crucial strong points. For instance, the first one is suitable for large-scale programs as its primary focus is on the semantic content of the code. The second one also has its benefits: this approach offers more possibilities in the field of structural and contextual analysis, which is vital for complex plagiarism checks. The key difference is that, unlike other systems, the proposed models can work with different programming code manipulations and coding styles, along with ensuring high accuracy of plagiarism detection.

The choice of the proposed implementations was driven by the need for a complex approach that contains the main strengths of an effective plagiarism detection system: adaptability to different plagiarism scenarios, precise document checking, and efficiency.

After implementing both proposed variants, a comparative analysis of the effectiveness of detecting various types of plagiarism in source code was conducted. The results of this analysis are presented in Table 3.

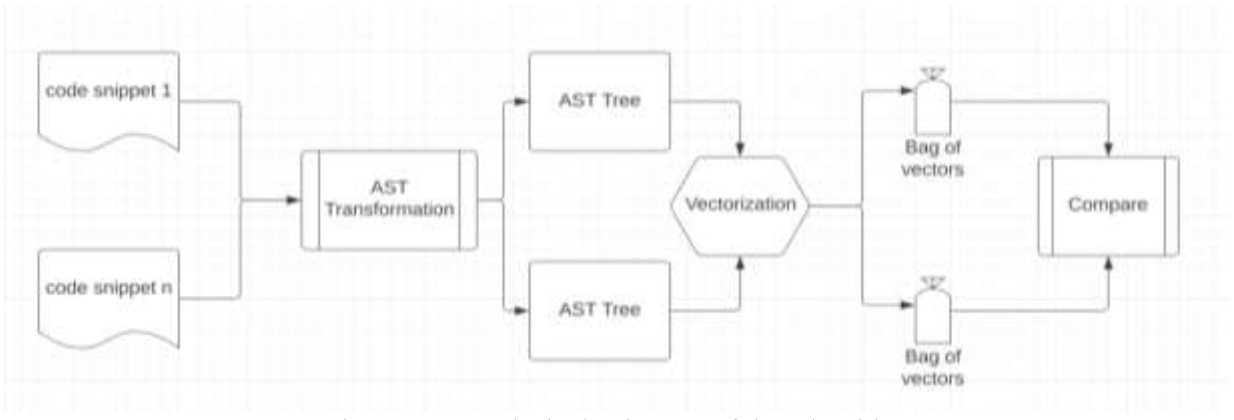

Fig. 2: Proposed Block Diagram of the Algorithm

Table 3. Results of the Comparative Analysis of the Proposed Solutions with the MOSS System

Type of plagiarism	TreeBERT	Multilayer perceptron + Code2Vec	MOSS
Identical source code fragments	100 %	100 %	100 %
Changed variable names and/or their types	95 %	94 %	100 %
Added or removed individual operators	95 %	95 %	92 %
Semantically equivalent code	79 %	85 %	64 %
Example of hash violation problem	93 %	98 %	10 %

It is important to note that the MOSS system is present in this experiment to evaluate the effectiveness of other proposed methods. It is also significant that all analyzed pairs of examples during this check contain plagiarism. Therefore, for maximum evaluation, which the system can provide, we consider 100%. A total of 5 types of situations were examined (4 of them demonstrate different types of program code plagiarism, and the 5th case represents a problem of hash collision). Compared to the MOSS system, neither developed method is sensitive to the hash collision problem. In cases where the MOSS system is vulnerable to this problem and shows a similar result of only 10%, the proposed methods achieve results of 93% and 98%, respectively. Thus, the developed models are better at detecting the fourth type of plagiarism, namely semantically equivalent code.

When comparing the two developed models, it can be concluded that the multilayer perceptron + Code2Vec has better performance indicators.

3.1 Limitations of the Study

One of the limitations of this study is the incomplete testing of the existing Codequiry system due to the absence of a free trial version of the system. Another limitation is the limited number of programming languages for the proposed systems, as the research and analysis of these systems' performance were conducted using a selection of code fragments in the Java programming language. Additionally, the study faced limitations related to the small size of the program code fragments used for training and testing the systems.

4 Discussion

Today, several systems and methods have been developed for detecting plagiarism, which helps identify fragments of program code that are plagiarized, and systems for checking program code that can detect semantically equivalent code. However, as numerical studies show, current systems for detecting program code plagiarism can very successfully identify identical program code and code with changed variable names and constants, and they can quite well identify code fragments with added or removed operators. The detection of semantically equivalent code is not as successful. Another issue is the vulnerability to hash collision attacks, which the system fails to recognize as plagiarism. This vulnerability is present in one of the popular systems for detecting plagiarism, MOSS, [28], [29], [30], [31], [32]. The work, [29], details the problem of hash collision and how the attack works. It was also proven in this work that it is possible to generate at least 12 variants of program code that, when input into the system, these variants will be semantically equivalent to the original, and MOSS will not detect plagiarism when comparing the original and one of the generated code variants. Also, according to work, [32], the MOSS system is one of the best systems for detecting program code plagiarism, as demonstrated by using various data selections and metrics like F-measures and precision-recall curves. Therefore, enhancing systems for detecting program code plagiarism is an important task, especially in detecting semantically equivalent code and the system's resistance to hash collision attacks. Research into existing systems and methods has proven their effectiveness in identifying identical fragments of program code. Methods and techniques for detecting plagiarism of semantically equivalent code require further refinement, [33]. This study focused on models and methods that have recently proven to be some of the most effective for detecting plagiarism in source code. Particular attention was paid to abstract syntax trees, [25], [26], neural networks, and the possibilities of their practical application in detecting program code plagiarism.

The models analyzed by us demonstrate very good results in the range of 79-85% for detecting semantically equivalent code and results of 93-95% in tests demonstrating the problem of hash collision, which strongly indicates their potential effectiveness in solving various tasks related to the detection of program code plagiarism. Thus, BERT, which was presented in the work, [34], and all subsequent modifications of this model are among the most promising for solving tasks related to the detection of program code plagiarism.

5 Conclusions and Perspectives for Further Research

During this research, two methods of checking the output program code for the presence of obfuscations were proposed – a multilayer perceptron + Code2Vec and TreeBert. For the TreeBert system, a function of checking for similarity was additionally developed, and, according to the authors, the main task of this model is code generation. In turn, the multilayer perceptron + Code2Vec was trained on a dataset containing 14 million examples of program code in Java. The designated code fragments from the selection were pre-processed by Code2Vec to detect similarities in vectors.

After conducting a comparative analysis, it was concluded that both proposed models resist the problem of hash collision, a major vulnerability of the existing MOSS system. It is also worth noting that the multilayer perceptron + Code2Vec shows better indicators than TreeBert and MOSS, especially in working with semantically equivalent code and in cases demonstrating the problem of hash collision. Since the method of this work was developing a more effective method for checking program code for plagiarism, the results of the work fully meet the set goals.

As a direction for further development, it is possible to highlight the optimization of the necessary time for conducting checks, working with a collection of documents, expanding the number of programming languages with which the system can work, and improving the system for analyzing large fragments of program code. To experiment with accuracy results and feasible results improvement, periodically checking the existing datasets is recommended. It is also possible to create personal datasets for further experiments.

References:

[1] Stephens, J. M. How to Cheat and not Feel Guilty: Cognitive Dissonance and its Amelioration in the Domain of Academic Dishonesty. *Theory into Practice*. Vol.56, No.2, 2017, pp. 111-120, https://doi.org/10.1080/00405841.2017.1283571.

[2] Sraka, D., & Kaučič, B., Source Code Plagiarism. Conference: Information Technology Interfaces. *Proceedings of the ITI 2009 31st International Conference*, Cavtat, Croatia, 2009, https://doi.org/10.1109/ITI.2009.5196127.

[3] Cosma, G., & Joy, M. *Source-Code Plagiarism: A UK Academic Perspective*, 2006, [Online]. https://www.researchgate.net/publication/228712855_Source-code_plagiarism_A_UK_academic_perspective (Accessed Date: October 23, 2023).

[4] Grimes, P. W. Dishonesty in Academics and Business: A Cross-Cultural Evaluation of Student Attitudes. *Journal of Business Ethics*. Vol.49, No.3, 2004, pp. 273-290, https://doi.org/10.1023/B:BUSI.0000017969.29461.30.

[5] Hard, S. F., Conway, J., & Moran, A. Faculty and College Student Beliefs about the Frequency of Student Academic Misconduct. *The Journal of Higher Education*. Vol.77, No.6, 2006, pp. 1058-1080, https://doi.org/10.1353/jhe.2006.0048.

[6] Cosma, G., & Joy, M. towards a Definition of Source-Code Plagiarism. *IEEE Transactions on Education*. Vol.51, No.2, 2008, pp. 195–200, https://doi.org/10.1109/TE.2007.906776.

[7] Joy, M., Cosma, G., Yin-Kim Yau, J., & Sinclair, J. Source Code Plagiarism – A Student Perspective. *IEEE Transactions on Education*. Vol.54, No.1, 2011, pp. 125–132, https://doi.org/10.1109/TE.2010.2046664.

[8] Baron, E. *Computer Science Cheating Incidents Part of Widespread Problem: Report*. Stanford, UC Berkeley, 2017.

[9] MacGregor, J., & Stuebs, M. To cheat or not to Cheat: Rationalizing Academic Impropriety. *Accounting Education*. Vol.21, No.3, 2012, pp. 265-287, https://doi.org/10.1080/09639284.2011.617174.

[10] Academic Cheating Fact Sheet. (1999) Educational Testing Service, [Online]. http://www.glass-castle.com/clients/www-nocheating-org/adcouncil/research/cheatingfactsheet.html (Accessed Date: October 22, 2023).

[11] McCabe, D. L., Butterfield, K. D., & Treviño, L. K. Cheating in college: Why students do it and what educators can do about it. Baltimore, Johns Hopkins University Press, 2012 [Online]. https://pure.psu.edu/en/publications/cheating-in-college-why-students-do-it-and-what-educators-can-do- (Accessed Date: October 23, 2023).

[12] Cheers, H., Lin, Y., & Smith, S. P. *Academic source code plagiarism detection by measuring program behavioural similarity*. arXiv, 2021, https://doi.org/10.48550/arXiv.2102.03995.

[13] Soeharno, J., Keimpe, A., & van der Schot, D. (Eds.). Plagiarism in Academic Research and Education. Hague, Association of Universities in the Netherlands, 2021, [Online]. https://www.researchgate.net/publication/354904254_Plagiarism_in_Academic_Research_and_Education (Accessed Date: October 23, 2023).

[14] Maryono, D., Yuana, R. A., & Hatta, P. The Analysis of Source Code Plagiarism in Basic Programming Course. Journal of Physics:

Conference Series, Vol.1193, 2019, paper 012027, https://doi.org/10.1088/1742-6596/1193/1/012027

[15] Zhao, G., & Huang, J. DeepSim: Deep learning code functional similarity. ESEC/FSE 2018: Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software. New York, Association for Computing Machinery, 2018, pp. 141–151, https://doi.org/10.1145/3236024.3236068.

[16] Husain, M. A., Wu, H.-H., Gazit, T., Allamanis, M., & Brockschmidt, M. CodeSearchNet Challenge: Evaluating the State of Semantic Code Search, 2020, https://doi.org/10.48550/arXiv.1909.09436.

[17] Alon, U., Zilberstein, M., Levy, O., & Yahav, E. code2vec: Learning Distributed Representations of Code, 2018, https://doi.org/10.48550/arXiv.1803.09473.

[18] Prechelt, L., Malpohl, G., & Philippsen, M. Finding Plagiarisms among a Set of Programs with JPlag. Journal of Universal Computer Science. Vol.8, No.11, 2003, pp. 1016-1038, [Online]. https://www.researchgate.net/publication/2832828_Finding_Plagiarisms_among_a_Set_of_Programs_with_JPlag (Accessed Date: October 23, 2023).

[19] Ahtiainen, A., Surakka, S., & Rahikainen, M. Plaggie: GNU-licensed Source Code Plagiarism Detection Engine for Java Exercises. Baltic Sea '06: Proceedings of the 6th Baltic Sea conference on Computing education research: Koli Calling. Uppsala, Sweden, 2006, pp. 141–142, https://doi.org/10.1145/1315803.1315831.

[20] Code Plagiarism & Similarity API. Codequiry, n.d., [Online]. https://codequiry.com/usage/api (Accessed Date: October 23, 2023).

[21] Simon, Karnalim, O., Sheard, J., Dema, I., Karkare, A., Leinonen, J., Liut, M., & McCauley, R. Choosing Code Segments to Exclude from Code Similarity Detection. Conference: ITiCSE-WGR '20: The Working Group Reports on Innovation and Technology in Computer Science Education, Trondheim, Norway, 2020, https://doi.org/10.1145/3437800.3439201.

[22] Compton, R., Frank, E., Patros, P., & Koay, A. Embedding Java Classes with code2vec: Improvements from Variable Obfuscation, 2020, [Online]. https://www.researchgate.net/publication/340499798_Embedding_Java_Classes_with_code2vec_Improvements_from_Variable_Obfuscation (Accessed Date: October 23, 2023).

[23] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. Attention is all You Need. *31st Conference on Neural Information Processing Systems*, Long Beach, CA, USA, 2017, [Online]. https://arxiv.org/pdf/1706.03762.pdf (Accessed Date: October 22, 2023).

[24] Jiang, X., Zheng, Zh., Lyu, Ch., Li, L., & Lyu, L. TreeBERT: A Tree-Based Pre-Trained Model for Programming Language. *37th Conference on Uncertainty in Artificial Intelligence*, Online, 2021, [Online]. https://arxiv.org/pdf/2105.12485.pdf (Accessed Date: October 23, 2023).

[25] Jones, J. *Abstract Syntax Tree Implementation Idioms*, 2003, [Online]. https://www.hillside.net/plop/plop2003/Papers/Jones-ImplementingASTs.pdf (Accessed Date: October 23, 2023).

[26] Tang, Z., Li, C., Ge, J., Shen, X., Zhu, Z., & Luo, B. *AST-Transformer: Encoding Abstract Syntax Trees Efficiently for Code Summarization*, 2021, [Online]. https://arxiv.org/pdf/2112.01184.pdf (Accessed Date: October 23, 2023).

[27] Kanade, A., & Maniatis, P., Balakrishnan, G., & Shi, K. *Learning and Evaluating Contextual Embedding of Source Code*, 2020, [Online]. https://arxiv.org/pdf/2001.00059v3.pdf (Accessed Date: October 23, 2023).

[28] Yang, D. *How MOSS Works*, 2019, [Online]. https://yangdanny97.github.io/blog/2019/05/03/MOSS (Accessed Date: October 22, 2023).

[29] Devore-McDonald, B., & Berger, E. D. Mossad: Defeating Software Plagiarism Detection. *Proceedings of the ACM on Programming Languages*, Vol. 4, No. OOPSLA, 2020, Article 1, [Online]. https://arxiv.org/pdf/2010.01700.pdf (Accessed Date: October 23, 2023).

[30] Moss, A System for Detecting Software Similarity. n.d, [Online]. https://theory.stanford.edu/~aiken/moss/ (Accessed Date: October 23, 2023).

[31] Schleimer, S., Wilkerson, D. S., & Aiken A. Winnowing: Local Algorithms for Document Fingerprinting, 2003, *SIGMOD '03: Proceedings of the 2003 ACM SIGMOD International Conference on Management of*

Data, June 2003, pp. 76–85, https://doi.org/10.1145/872757.872770.

[32] Heres, D., & Hage, J. A Quantitative Comparison of Program Plagiarism Detection Tools. *Conference: the 6th Computer Science Education Research Conference*. New York, NY, USA, 2017, pp. 73-82, https://doi.org/10.1145/3162087.3162101.

[33] Karnalim, O., Simon, & Chivers, W. J. Preprocessing for Source Code Similarity Detection in *Introductory Programming. Conference: 20th Koli Calling International Conference on Computing Education Research*, Koli, Finland, 2020, pp. 1-10, https://doi.org/10.1145/3428029.3428065.

[34] Arase, Y., & Tsujii, J. 2021. Transfer fine-tuning of BERT with phrasal paraphrases. *Computer Speech & Language*. Vol.66, 2021, paper 101164, https://doi.org/10.1016/j.csl.2020.101164.

Contribution of Individual Authors to the Creation of a Scientific Article (Ghostwriting Policy)

The authors equally contributed in the present research, at all stages from the formulation of the problem to the final findings and solution.

Sources of Funding for Research Presented in a Scientific Article or Scientific Article Itself

No funding was received for conducting this study.

Conflict of Interest

The authors have no conflicts of interest to declare.