Acceleration, Quantum, Information Technology and Algorithm Journal (AQILA)
Vol. 3, No. 1, June 2026, pp 29-34
ISSN: 3062-8555, DOI: 10.62123/aqila.v3i1.177

Code Plagiarism Detection Using Graphic Neural Network Based On Abstract Syntax Tree

Fitra Affandi Hasibuan[1], Al-Khowarizmi[1*]

[1]Department of InformationTechnology, Universitas Muhammadiyah Sumatera Utara, Indonesia

*Corresponding Email: alkhowarizmi@umsu.ac.id

DOI : 10.6213/aqila.v3i1.177

Received : May 31, 2026
Revised : June 15, 2026
Accepted : June 16, 2026

Keywords:
Code Plagiarism
Graph Neural Network
Abstract Syntax Tree
Cosine Similarity
Similarity Detection

ABSTRACT

Code plagiarism is a common issue in education and software development, which is difficult to detect accurately using text-based approaches. Conventional methods such as Term Frequency–Inverse Document Frequency (TF-IDF) and cosine similarity tend to focus only on token similarity, making them less effective in handling structural changes in code. Therefore, this study aims to develop a structure-based code plagiarism detection system using Abstract Syntax Tree (AST) and Graph Neural Network (GNN). The proposed method involves parsing source code into AST, representing it as a graph, and processing it using a GNN model in a pairwise scheme. In addition, a comparison is conducted with a baseline method based on TF-IDF and cosine similarity to evaluate model performance. The dataset used consists of both synthetic and real data, which are divided into training and testing sets. The results show that the GNN model achieves excellent performance with an accuracy of 0.9946, precision of 0.9949, recall of 0.9974, and F1-score of 0.9962, while the baseline method only achieves an accuracy of 0.7392 and a recall of 0.6343. These results indicate that the GNN model is more effective in detecting plagiarism, especially in handling structural code modifications. Therefore, it can be concluded that the structure-based approach using AST and GNN outperforms text-based approaches in code plagiarism detection.

1. INTRODUCTION

The development of information technology has made programming a fundamental competency in higher education, particularly in computer science. Programming assignments are often used as a primary instrument in the learning evaluation process. However, easy access to various program code sources on the internet opens up opportunities for plagiarism among students. Plagiarism is no longer limited to direct copying, but involves various modifications such as variable renaming, changes in writing structure, comment removal, and minor refactorings aimed at avoiding detection [1].

Most program code plagiarism detection systems still use text-based approaches, such as token comparison, TF-IDF [2], and cosine similarity [3]. These approaches only measure surface similarity and are very sensitive to small changes in code writing, making them less effective in detecting plagiarism that has been structurally modified [2][4][5]. To address this issue, an approach capable of explicitly representing the program's syntactic structure is needed. An Abstract Syntax Tree (AST) is a structural representation of program code that can illustrate hierarchical relationships between syntax elements regardless of the writing format [6]. AST representation allows the system to detect similarities in program logic even when variable names or writing formats are changed [7].

This study proposes a structure-based program code plagiarism detection approach by utilizing an Abstract Syntax Tree, converted into a graph, and then processed using a Graph Neural Network (GNN) [8][9]. GNNs are able to learn the relationship patterns between nodes in the graph through a message-passing mechanism, thus producing an embedding representation that reflects the characteristics of the program code structure [10]. With this approach, the system is expected to be able to detect structure-based plagiarism more accurately than text-based methods [11].

2. RESEARCH METHODOLOGY

This research develops a program code plagiarism detection system using a pairwise comparison scheme [12], which compares two pieces of program code, code A and code B, to determine the level of structural similarity and plagiarism status

[13][14]. The system flow begins with source code preprocessing, generating a structural representation using an Abstract Syntax Tree (AST), converting the AST into a graph, and then processing the graph using a Graph Neural Network (GNN) model [15] [16].

In the initial stage, the source code is cleaned of elements that do not affect the syntactic structure, such as comments and excess whitespace. This process aims to ensure that the analysis focuses strictly on the program's logical structure. The cleaned code is then parsed using a Python parser to form an Abstract Syntax Tree (AST) [17]. An AST represents the program's syntactic structure as a node hierarchy that illustrates the parent-child relationships between code elements. The resulting AST is then converted into a graph by mapping each AST node as a graph node, and each parent-child relationship as an edge. This graph representation allows the relationships between program syntax elements to be analyzed as relational structures, rather than simply as sequences of tokens.

The AST graphs of code A and code B are then processed using a Graph Neural Network (GNN). Through a message-passing mechanism, each node in the graph repeatedly aggregates information from its neighboring nodes across multiple layers. This process produces a graph-level embedding representation that reflects the overall structural characteristics of the program code. The embeddings from both graphs are compared to generate a similarity score. This similarity score is then compared with a specific threshold to determine whether the code pair falls into the plagiarized category.

The dataset used in this study is structured as a pairwise dataset consisting of two categories: plagiarized pairs and non-plagiarized pairs. Plagiarized pairs are formed through directed modifications to the original code without altering the program logic, such as changing variable names, changing the writing format, and deleting comments. Meanwhile, non-plagiarized pairs are derived from two code pieces that are logically unrelated. The GNN model is trained using training data, validated using validation data, and evaluated using test data. Model performance evaluation was conducted using accuracy, precision, recall, and F1-score metrics to measure the system's ability to detect structure-based program code plagiarism [18][19].

3. RESULTS AND DISCUSSION

3.1. System Implementation

In this study, a program code plagiarism detection system was implemented using a structure-based approach utilizing an Abstract Syntax Tree (AST) and a Graph Neural Network (GNN) [14]. The system was designed to compare two Python program codes and generate a similarity score as a basis for determining the level of plagiarism.

The detection process begins with input of two program codes [20], which are then parsed into an AST. The AST structure is then converted into a graph and processed using a Siamese Network-based GNN model to generate an embedding representation of each code. These embeddings are then compared to obtain a similarity score and plagiarism label. The developed system architecture is shown in Figure 1.

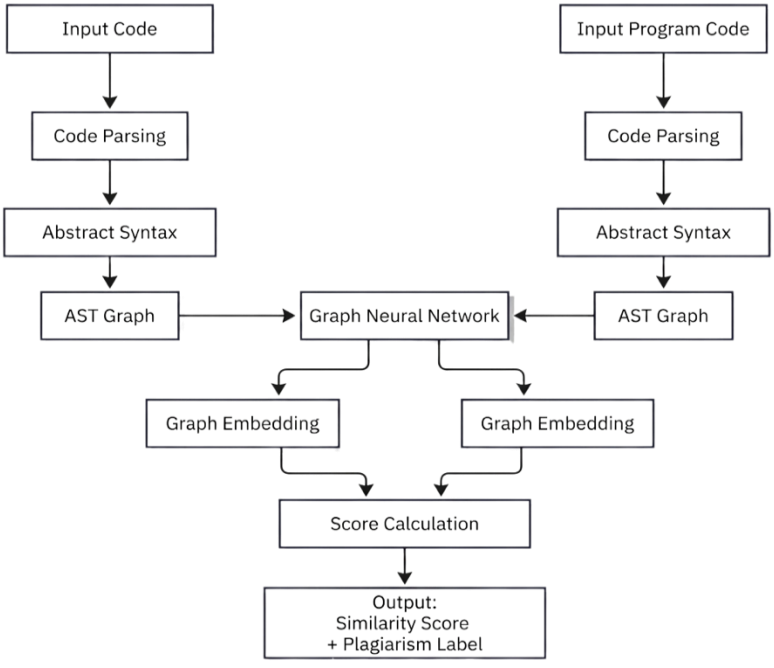

Figure 1. Architecture of AST and GNN-Based Program Code Plagiarism Detection System

Based on Figure 1, the system works through several main stages: code parsing, AST formation, graph AST conversion, embedding using GNN, and similarity score calculation. The resulting similarity score is then interpreted into low, medium, and high plagiarism categories. Furthermore, the system is equipped with a GUI-based user interface using Tkinter to simplify the testing process. The system's main GUI display is shown in Figure 2.

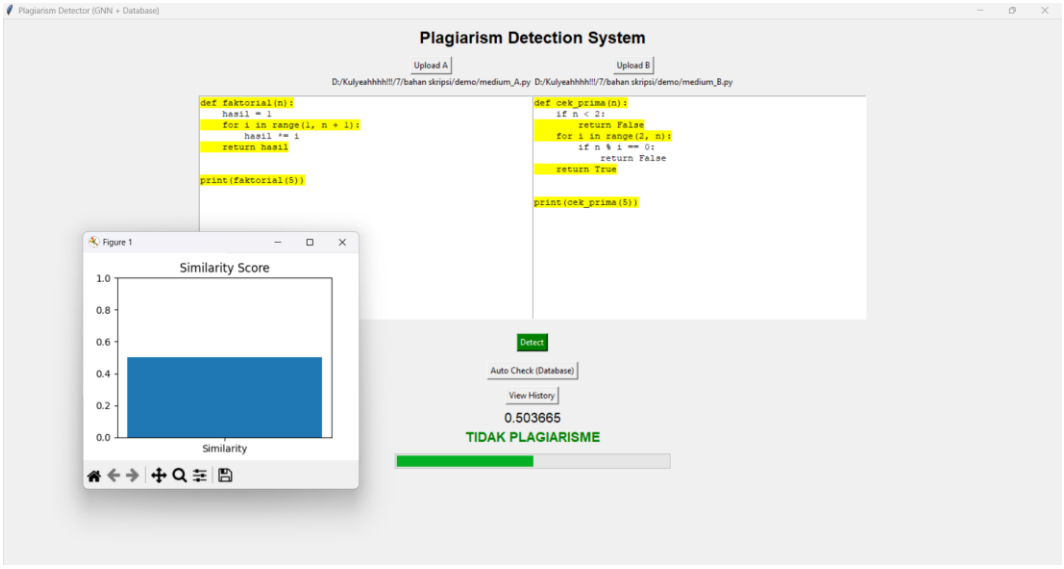

Figure 2. Main Interface (GUI) of the Plagiarism Detection System

3.2. Dataset Distribution

The dataset used in this study consists of program code pairs that have been labeled plagiarized and non-plagiarized. The dataset distribution is shown in Table 1.

Table 1. Dataset Distribution

Category	Label	Amount of Data
Plagiarisme	1	978
Non-Plagiarisme	0	326
Total		1034

Table 1 shows that the dataset distribution is unbalanced, with plagiarized data outnumbering non-plagiarized data. This imbalance was intentionally maintained to represent the real-world conditions of student programming assignments, where plagiarized variations are more common than truly different code. The dataset was then randomly divided into training data and testing data in an 80:20 ratio. This division aims to train the model and evaluate its ability to generalize to previously unseen data. The dataset distribution is shown in Table 2.

Table 2. Dataset Distribution

Dataset Type	Amount of Data
Training	1034
Testing	261
Total	1304

3.3. Model Evaluation Results

Model evaluation was conducted to measure the ability of a Graph Neural Network (GNN) to detect plagiarism in program code based on its Abstract Syntax Tree (AST) structural representation. Testing was conducted using test data not used during the training process so that the evaluation results could demonstrate the model's generalization ability to new data.
Model performance was measured using several evaluation metrics, namely accuracy, precision, recall, and F1-score. The model evaluation results are shown in Table 3.

Table 3. GNN Model Evaluation Results

Metric	Value
Accuracy	0.9946
Precision	0.9949
Recall	0.9974
F1-Score	0.9962

Based on Table 3, the model demonstrated excellent performance across all evaluation metrics. An accuracy value of 0.9946 indicates that the model was able to correctly classify most code pairs. The high precision value indicates that the model's plagiarism predictions had a very high level of accuracy [21]. Furthermore, a recall value of 0.9974 indicates that the model was able to detect almost all cases of plagiarism in the dataset. This is crucial in a plagiarism detection system because false negative errors need to be minimized. The high F1-score also indicates that the model strikes a good balance between precision and recall. These evaluation results demonstrate that the structure-based approach using AST and GNN is able to effectively detect code plagiarism, including in cases involving variable name changes and modifications to code formatting.

3.4. Results and Case Analysis

After the model is trained, testing is performed using test data to determine its ability to detect various cases of program code plagiarism. At this stage, the model generates a similarity score, which is used to determine the level of similarity between two pieces of code. In general, the Graph Neural Network (GNN) model is capable of providing high similarity scores for plagiarized code pairs and low scores for non-plagiarized code pairs. Examples of model testing results are shown in Table 4.

Table 4. Example of Model Testing Results.

No	Original Label	GNN Score	Baseline Score	GNN Prediction	Baseline Prediction
1.	1	0.9912	1.0000	1	1
2.	1	0.9934	0.7848	1	1
3.	1	0.9879	0.3002	1	0
4.	1	0.9795	0.2330	1	0
5.	1	0.9899	0.2101	1	0
6.	1	0.9911	0.8876	1	0
7.	0	0.0000	0.1133	0	0
8.	0	0.0000	0.0213	0	0
9.	0	0.0002	0.1100	0	0
10.	0	0.0001	0.0102	0	0

Based on Table 4, the GNN model produces consistent similarity scores for both plagiarized and non-plagiarized data. For plagiarized code pairs, the model produces high similarity scores, while for non-plagiarized code pairs, the scores are very low. This indicates that the model is able to distinguish between the two categories well. Compared to baseline methods based on TF-IDF and cosine similarity, the GNN model demonstrates more stable performance, especially in cases of modified plagiarism. The baseline method tends to fail to detect some cases of plagiarism because it focuses only on textual similarities, while the GNN model is able to understand the relationship between code structures through AST representations.

Further analysis shows that the model performs very well in cases of high plagiarism, when two codes have very similar structures. Furthermore, the model is also able to detect cases of medium plagiarism even when there are changes in variable names, writing formats, or simple modifications to the program code. However, in certain cases, the model still produces high similarity scores for code with similar structures but different program logic. This indicates that the model focuses more on structural similarity than on a deeper understanding of the code's semantics [22]. Overall, the test results show that the AST- and GNN-based approaches are effective in detecting code plagiarism and outperform text-based methods.

3.5. Comparison with Baseline Method

This study compares a Graph Neural Network (GNN) model based on an Abstract Syntax Tree (AST) with a baseline method using TF-IDF and cosine similarity. The comparison was conducted to determine the effectiveness of the structure-based approach compared to the text-based approach in detecting program code plagiarism. The results of the performance comparison of the two methods are shown in Table 5.

Table 5. Model Performance Comparison

Metode	Accuracy	Precision	Recall	F1-Score
GNN	0.9946	0.9949	0.9974	0.9962
TF-IDF plus Cosine Similarity	0.7392	0.9940	0.6343	0.7744

Based on Table 5, the GNN model performed better than the baseline method across all evaluation metrics. The most significant difference was seen in the recall value, where the GNN model was able to detect almost all cases of plagiarism, while the baseline method still failed to detect some cases that had been modified. These results indicate that the structure-based approach using AST and GNN is more effective than the text-based approach in detecting plagiarized program code

3.6. Discussion

The research results show that the Abstract Syntax Tree (AST) and Graph Neural Network (GNN)-based approaches are capable of detecting plagiarism in program code with excellent performance. The AST representation allows the system to understand the program's syntactic structure, while the GNN is able to learn the relationships between nodes in the graph in greater depth. Based on evaluation and testing results, the model is capable of detecting various forms of plagiarism, including cases where the code has been modified through variable name changes, comment deletions, and formatting changes. This demonstrates that the structure-based approach is more effective than text-based methods that focus solely on token similarity.

Furthermore, comparisons with baseline methods indicate that the GNN model has superior performance, particularly in terms of recall and F1-score. This demonstrates that utilizing the graph representation of the AST can improve the system's ability to recognize similar patterns in program code. However, the model still has limitations in some cases where code has a similar structure but different program logic. This suggests that the model still focuses more on structural similarity than on a comprehensive understanding of the code's semantics. Overall, this study proves that the combination of AST and GNN is an effective approach in detecting program code plagiarism and is able to overcome the limitations of conventional text-based methods.

4. CONCULUSSION

Based on the research results, it can be concluded that a program code plagiarism detection model based on an Abstract Syntax Tree (AST) and a Graph Neural Network (GNN) has been successfully developed and implemented in a pairwise scheme (code A vs. code B). The developed system is capable of generating similarity scores and classifying plagiarism automatically and objectively. The GNN model demonstrated excellent performance in detecting program code plagiarism, particularly in Type 1 (cosmetic copy-paste) and Type 2 (identifier renaming) cases. This is evidenced by high accuracy, precision, recall, and F1-score values on the test data, indicating that the model is able to effectively recognize patterns of code structure similarity.

Compared to baseline methods based on TF-IDF and cosine similarity, the GNN model demonstrated superior performance. The most significant difference was seen in the model's ability to detect plagiarism cases involving surface code changes, such as variable renaming. This indicates that the structure-based approach using AST and GNN is more effective than the text-based approach. Furthermore, the model demonstrated good ability to distinguish between plagiarized and non-plagiarized code based on program structure. However, in some cases, the model still experienced limitations in distinguishing code with similar structures but different logic, indicating that the semantic understanding of the code is not yet fully optimized. Overall, this study demonstrates that utilizing AST representations combined with Graph Neural Networks (GNNs) is an effective approach for detecting plagiarized program code. This approach overcomes the limitations of conventional text-based methods and provides more accurate and robust results under various testing conditions.

Based on the research findings, several suggestions can be considered for further research.

1. Model development should be directed at improving the ability to understand the semantic aspects of program code. This is crucial to address the limitations of models that tend to focus on structural similarities, thus enabling them to distinguish code with similar structures but different logic.
2. A larger, more diverse, and balanced dataset is needed to improve the model's generalizability. Furthermore, adding more complex plagiarism types, such as Type-3 and Type-4, could provide a more comprehensive evaluation of system performance under conditions more closely resembling real-world conditions.
3. Further research is recommended to test and develop the model in various programming languages besides Python. This aims to determine the model's adaptability to differences in syntax and structure in other programming languages, thereby increasing the system's flexibility and usability.
4. System development can be directed at integration with learning platforms or learning management systems (LMS), so that it can be directly utilized as a tool in the automated and objective evaluation of programming assignments.
5. Exploration of approaches that combine structural and semantic representations, such as embedding-based models or hybrid approaches, can be an alternative to improve the accuracy and robustness of the system in detecting program code plagiarism.

REFRENCES

[1] M. Zakeri-Nasrabadi, S. Parsa, M. Ramezani, C. Roy, and M. Ekhtiarzadeh, "A systematic literature review on source code similarity measurement and clone detection: Techniques, applications, and challenges," *Journal of Systems and Software*, vol. 204, p. 111796, Oct. 2023, doi: 10.1016/j.jss.2023.111796.

[2] I. G. A. E. Putra and I. W. Supriana, "Deteksi Plagiarisme Source Code Tugas Mahasiswa Menggunakan Algoritma Cosine Similarity Dan Pembobotan TF-IDF," *J. Nas. Teknol. Inf. dan Apl*, vol. 1, no. 1, p. 575, 2022.

[3] Z. Zhang and T. Saber, "Exploring the Boundaries Between LLM Code Clone Detection and Code Similarity Assessment on Human and AI-Generated Code," *Big Data and Cognitive Computing*, vol. 9, no. 2, p. 41, Feb. 2025, doi: 10.3390/bdcc9020041.

[4] R. Maertens *et al.*, "Discovering and exploring cases of educational source code plagiarism with Dolos," *SoftwareX*, vol. 26, p. 101755, May 2024, doi: 10.1016/j.softx.2024.101755.

[5] Z. Dong, Q. Hu, Z. Zhang, and J. Zhao, "On the effectiveness of graph data augmentation for source code learning," *Knowl. Based. Syst.*, vol. 285, p. 111328, Feb. 2024, doi: 10.1016/j.knosys.2023.111328.

[6] O. O. Büyük and A. Nizam, "Deep learning with class-level abstract syntax tree and code histories for detecting code modification requirements," *Journal of Systems and Software*, vol. 206, p. 111851, Dec. 2023, doi: 10.1016/j.jss.2023.111851.

[7] X. Guo and J. Ma, "Heritage applications of landscape design in environmental art based on image style migration," *Results in Engineering*, vol. 20, p. 101485, Dec. 2023, doi: 10.1016/j.rineng.2023.101485.

[8] J. Zhou *et al.*, "Graph neural networks: A review of methods and applications," *AI Open*, vol. 1, pp. 57–81, 2020, doi: 10.1016/j.aiopen.2021.01.001.

[9] C. Chen *et al.*, "A Survey on Graph Neural Networks and Graph Transformers in Computer Vision: A Task-Oriented Perspective," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 46, no. 12, pp. 10297–10318, Dec. 2024, doi: 10.1109/TPAMI.2024.3445463.

[10] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu, "A Comprehensive Survey on Graph Neural Networks," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 32, no. 1, pp. 4–24, Jan. 2021, doi: 10.1109/TNNLS.2020.2978386.

[11] Y. Wu and J. Wan, "A survey of text classification based on pre-trained language model," *Neurocomputing*, vol. 616, p. 128921, 2025.

[12] C. K. Roy, J. R. Cordy, and R. Koschke, "Comparison and evaluation of code clone detection techniques and tools: A qualitative approach," *Sci. Comput. Program.*, vol. 74, no. 7, pp. 470–495, May 2009, doi: 10.1016/j.scico.2009.02.007.

[13] C. Li, A. Sirikham, J. Konpang, and Y. Wang, "Code Clone Detection With Self-Supervision on Dual Graphs," *Operational Research in Engineering Sciences: Theory and Applications*, vol. 7, no. 4, 2024.

[14] J. Guo, J. Liu, X. Liu, Y. Wan, and L. Li, "Summarizing source code with Heterogeneous Syntax Graph and dual position," *Inf. Process. Manag.*, vol. 60, no. 5, p. 103415, Sep. 2023, doi: 10.1016/j.ipm.2023.103415.

[15] Y. Zhang, J. Yang, and O. Ruan, "Cross-language Source Code Clone Detection Based On Graph Neural Network," in *Proceedings of the 2024 3rd International Conference on Cryptography, Network Security and Communication Technology*, New York, NY, USA: ACM, Jan. 2024, pp. 189–194. doi: 10.1145/3673277.3673310.

[16] Q. Yu, X. Liu, Q. Zhou, J. Zhuge, and C. Wu, "Code classification with graph neural networks: Have you ever struggled to make it work?," *Expert Syst. Appl.*, vol. 233, p. 120978, Dec. 2023, doi: 10.1016/j.eswa.2023.120978.

[17] G. Yang, T. Jin, and L. Dou, "Heterogeneous Directed Hypergraph Neural Network over abstract syntax tree (AST) for Code Classification," Jul. 2023, pp. 274–279. doi: 10.18293/SEKE2023-136.

[18] F. Ebrahim and M. Joy, "Source Code Plagiarism Detection with Pre-Trained Model Embeddings and Automated Machine Learning," in *Proceedings of the Conference Recent Advances in Natural Language Processing - Large Language Models for Natural Language Processings*, INCOMA Ltd., Shoumen, BULGARIA, 2023, pp. 301–309. doi: 10.26615/978-954-452-092-2_034.

[19] F. Ebrahim and M. Joy, "Semantic Similarity Search for Source Code Plagiarism Detection: An Exploratory Study," in *Proceedings of the 2024 on Innovation and Technology in Computer Science Education V. 1*, New York, NY, USA: ACM, Jul. 2024, pp. 360–366. doi: 10.1145/3649217.3653622.

[20] H. Yang, Z. Li, and X. Guo, "A Novel Source Code Clone Detection Method Based on Dual-GCN and IVHFS," *Electronics (Basel).*, vol. 12, no. 6, p. 1315, Mar. 2023, doi: 10.3390/electronics12061315.

[21] Z. Li, H. Lei, Z. Ma, and F. Zhang, "Code Similarity Prediction Model for Industrial Management Features Based on Graph Neural Networks," *Entropy*, vol. 26, no. 6, p. 505, Jun. 2024, doi: 10.3390/e26060505.

[22] D. Yu, Q. Yang, X. Chen, J. Chen, and Y. Xu, "Graph-based code semantics learning for efficient semantic code clone detection," *Inf. Softw. Technol.*, vol. 156, p. 107130, Apr. 2023, doi: 10.1016/j.infsof.2022.107130.