A tool for detecting similarity in Jupyter Notebooks used as assessment reports

Nikesh Bajaj, Dimitrios Chiotis, Reza Moosaei, Jordan B. L. Smith, Pengfei Fan, Jesús Requena Carrión
Data Science and Engineering Research Group, School of Physical and Chemical Sciences
Queen Mary University of London, London, UK
{nikesh.bajaj, d.chiotis, r.moosaei, jordan.smith, pengfei.fan, j.requena}@qmul.ac.uk

***Abstract*—Coding-based assessment is essential not only for evaluating students' programming skills but also for assessing their ability to apply theoretical concepts to real-world scenarios. Jupyter Notebooks constitute a unique platform to assess coding-based pieces of work. Using a Jupyter Notebook, students can write, execute and visualise code while documenting their process through explanations, equations, and multimedia elements. However, identifying cases of plagiarism and collusion in Jupyter Notebooks can be challenging. Since Jupyter Notebooks cannot be processed by traditional text-based plagiarism detection tools, the ability to identify plagiarism largely relies on manual approaches. Hence, to fully benefit from Jupyter Notebooks as an assessment platform, automatic approaches that can contribute to identifying plagiarism and collusion are needed. In this paper, we introduce a new tool - *JBEval*, that identifies similar code and text blocks in a collection of Jupyter Notebooks and provides similarity scores for overall code and text individually. *JBEval* also provides a visual comparison of any two Jupyter Notebooks, allowing to trace identical blocks for further investigation and reporting. Our similarity detection algorithm is parameterized for controlling the aggressiveness of the similarity-based detection process, which makes it adaptable to a wide diversity of courses. This fast and reliable tool will be made widely accessible by sharing it in the public domain, enabling educators across diverse disciplines to integrate it into their assessment workflows.**

***Index Terms*—Jupyter Notebooks, Classroom Assessment, Plagiarism, Collusion, Similarity, Visualisation**

I. Introduction

The primary objective of assessments in education is to evaluate if the intended learning outcomes have been achieved for a given course and for the overall program. Assessments also helps educators to check the progress of students and to identify potential areas where students might need additional support. With the rapid development of digital technologies and their application in a wider range of fields, programs have started to incorporate coding teaching in their curricula and adding coding skills as learning outcomes.

The role of coding-based assessment is to evaluate students' programming skills. In recent years, the use of auto-grading tools in coding-based assessment has consolidated, providing student instant feedback on the correctness of the code [1], [2]. However assessing for only code correctness is not always desirable. Coding-based pieces of assessment can also be designed to evaluate higher-level skills that include being able to apply theoretical concepts to real-world scenarios, to explore data, and to carry out critical analysis. For such assessments, coding steps can vary among students and there are no unique answers, rendering auto-graders ineffective. This form of assessment can mimic real-world scenarios which students are likely to face while working as professionals. Assessment mimicking real-world scenarios in education setting are known as *Authentic Assessment* [3].

Jupyter Notebooks provide a unique platform for educators to create forms of assessment which combine coding, writing text and including multimedia components in a single report. This allows students to execute code, explore the implemented approach, add visualisations and document the entire process with text, equations and multimedia elements in one single file. This integration of coding and report writing enhances the learning experience by allowing students to explain their code, interpret results, and present insights in a cohesive manner - like telling a story. It fosters a deeper understanding of both the technical and conceptual aspects of a task, promoting the development of well-rounded problem-solving, analytical, and communication skills. The pedagogical importance of incorporating Jupyter Notebooks in teaching has already been established [4]–[6] and educators across disciplines are increasingly adapting Jupyter Notebooks into their practice [6], [7].

However, the use of Jupyter Notebooks for assessment poses as a challenge that of identifying instances of plagiarism and collusion, which go violate academic integrity. The widespread availability of code on the web, combined with its potential limitless distribution among students, make this form of assessment particularly prone to cases of academic misconduct. Traditional plagiarism detection tools are not compatible with reports using Jupyter Notebooks, which leaves assessors with the sole responsibility to identify academic misconduct cases using manual approach. Manual approaches could be effective in some extent to detect the cases of collusion (unfair collaboration). However, detecting plagiarism (e.g. comparing against previous cohort) is significantly more challenging.

In this paper, we introduce a new tool - *JBEval*, that is designed to process a collection of Jupyter Notebooks and detect both similar code and text blocks among them. Given two reports A and B written as a Jupyter-Notebook format), *JBEval* produces a overall similarity score indicating the amount of code in A is similar to code in B. Furthermore, an overall similarity score for text blocks is computed. *JBEval* also provides a visual representation for the comparison between two reports, which aids assessors and educators to

further investigate collusion and plagiarism by tracing the identified blocks.

Our algorithm implemented in *JBEval* is parameterised for controlling the aggressiveness of similarity. It allows educators to tune the level of similarity between two strings, the minimum size of a block, and the acceptable overall similarity score threshold. These tunable parameters make *JBEval* adaptable to a wide diversity of courses. We also aspire to substantially contribute to the accessibility of our tool by making it publicly available to the education community. We have developed a cloud-hosted application for *JBEval* and plan to release it to the public.

II. Background

The conciseness of the definition of plagiarism can lead us to think that the students' perception of plagiarism is aligned with this definition. A recent study conducted by Al-Hashmi et. al., [8] investigated whether this is the case. Hashmi's study revealed that the majority of students have a general idea of what plagiarism is, however they lack the understanding of what it implies, suggesting the need for more awareness around plagiarism, even beyond an educational setting [9]. One of the most common contributors to widespread plagiarism has been suggested to be the easy access to online resources [8]. Specifically, in the case of coding-based pieces of assessment, the ease of reusing code has been highlighted in relation to plagiarism [10].

Tools for plagiarism and collusion detection for coding-based forms of assessment have recently gained increased popularity in the education community, as coding skills have been increasingly added as learning outcomes in diverse programs and courses. Several tools have been created to support the education community and they vary in terms of their target language, compatibility, flexibility and availability.

Early work in this area focused on computing similarity between two code blocks mainly targeting languages such as C [11] Pascal, Unix shell and Fortran [12]. These approaches are mostly based on processing the code blocks by cleaning and transformations to build new meaningful entities to compare and compute similarity scores between the two. MOSS, which stands for measure of software similarity [13], is another tool first introduced in 1994 which currently supports a large number of programming languages and is available to use through online services. MOSS is based on a fingerprinting algorithm designed for documents.

Among the new approaches focused on plagiarism detection, JPlag is a popular tool that works with source code [14]. Currently, JPlag supports a large number of programming languages, similar to MOSS. Other similarity scores based on clustering approaches have also been proposed [15].

Most of the code similarity tools require code-only files and have a fix setting for similarity detection, which makes them useful for particular fields and types of assessments. There exist paid services for code similarity detection, some of which are based on these open-source platforms. However, the details of the particular approach implemented in paid-service are not usually shared for protection reasons.

Despite the availability of tools, there is a need for an open-source, parametrised tool that is flexible enough to suit the requirements of different forms of assessment in courses from different fields and different levels. Specifically, there is a need for a tool that supports Jupyter Notebooks directly and processes them as a single report, where there might be fragments of code and text.

III. Methods

A Jupyter Notebook organises all its contents as cells, which appear as sequential blocks of contents. There are three main types of cells in the Jupyter Notebook JSON structure, namely *markdown*, *raw*, and *code*. Markdown contents are written using HTML style, raw cells includes plain text and code cells capture all the code blocks.

A. Code and Plain-Text Extraction

After extracting the contents from a Jupyter Notebook as cells, we separate them into two categories, code-blocks and raw-text. All the code blocks are cleaned and concatenated in order to make one single code component C_c. Similarly, we clean and process plain text from raw cells and make one single plain text component as C_p. We leave the Markdown cells, assuming that they include the instructions provided in the assessment.

B. Preprocessing - cleaning

The code component C_c, is cleaned by removing empty lines and comments. In the Python language all the comments start with the symbol '#', which makes it suitable to extract only code lines, even in cases where a single line content code is followed by a comment. As the objective of processing code component is to find similarity, we further process it by removing extra spaces and indentation, leaving only code segments as sequences of lines. For the purpose of cleaning raw text we do not remove any text. We do remove however extra spaces and segment text into lines.

C. Similarity detection and score

This section explains the similarity detection process and the computation of our similarity score .

1) String matching: The similarity between two code strings c_1 and c_2 is computed using Jaro-Winkler's similarity $S \in [0, 1]$ [16]. We set a threshold T for S, to decide whether two code strings are similar or not. For example, if $S_{c_1,c_2} > T$, we consider that code strings c_1 and c_2 are significantly similar. The selection of the threshold T allows us to control the aggressiveness of the similarity detection algorithm. Choosing a higher value for the threshold T will eliminate the cases of similarity where two lines of code are identical except for a small variation. To illustrate Jaro-Winkler similarity score, consider the strings c_1 = '*for i in range(10):*, c_2 = '*for k in range(10):*', c_3 = '*for k in LIST_10:*' and c_4 = '*for k in LIST_A:*'. The similarity between c_1 and c_2,

c_3 and c_4 is, respectively, $S_{c_1,c_2} = 0.9678$, $S_{c_1,c_3} = 0.8270$ and $S_{c_1,c_4} = 0.6417$.

2) Block matching: When programming frequent tasks, such as plotting figures, reading files and importing libraries, are encoded using very similar lines of code irrespective of the project. This can result in a higher number of matching lines between the code produced by two students.

In order to overcome the many instances of single-line matching, we adopt a block-matching approach by identifying a block of lines in report A which is similar to a block of lines in a report B. We consider a block b_i in report A to be similar to another block b_j in report B if a minimum number of consecutive lines L_m match, in other words, if each of $L \geq L_m$ lines of code in b_i is similar ($S > T$) to another corresponding line in b_j. A collection of blocks identified in this manner are denoted by P.

3) Non-overlapping matching blocks: Once we obtained all the matched blocks, we filter out (remove) the blocks which are matched multiple times and overlapping blocks by keeping the largest ones. This is done to ensure, we count a matched line only once. As a results, we obtain a collection of non-overlapping blocks, which we denote as Q.

4) Overall similarity score: After identifying all the non-overlapping blocks $b_j \in Q$ in A that are similar to blocks in B, we finally compute an overall similarity score $\textbf{\textit{स}}_{A \leftarrow B}$ (where the Devanagari symbol स is pronounced as 'sə'), as the fraction of code in A, that is similar to code in B

$$\textbf{\textit{स}}_{A \leftarrow B} = \frac{\sum_j |b_j|}{|C_c|} = \frac{N_{A \leftarrow B}}{N_A} \tag{1}$$

where $N_{A \leftarrow B} = \sum_j |b_j|$ is the total number of lines in non-overlapping blocks from A that are similar to code lines in B, and $N_A = |C_c|$ is total number of code lines in A. The similarity score $\textbf{\textit{स}}_{A \leftarrow B}$ offers the following interpretations:

- **Student specific**: $\textbf{\textit{स}}_{A \leftarrow B}$ quantifies the fraction of A that is similar to B. If report A consists of 100 lines of code and a match of 50 lines of code is found, then 50% of A is similar to B, regardless of the number of lines in B (i. e. 500 lines of code).
- **Asymmetry**: $\textbf{\textit{स}}_{A \leftarrow B}$ is not the same as $\textbf{\textit{स}}_{B \leftarrow A}$. This means that the fraction of work that student A has similar to student B, is not the same as the fraction that student B has similar to A. This could be the result of partial collusion/plagiarism, where students only worked on partial problems together. Alternatively, this could be the result of student A, managing to acquire only partial work from B. In both cases, the asymmetry of $\textbf{\textit{स}}_{A \leftarrow B}$ can give insights into the nature of the collusion/plagiarism.

D. Processing entire cohort

As the entire procedure is implemented as software, analysing an entire cohort together and comparing every student with every other within a cohort can be done quickly. The procedure implemented also allows for comparison of the reports from students from a current cohort to all the previous cohorts. Once all the reports are analysed, a threshold

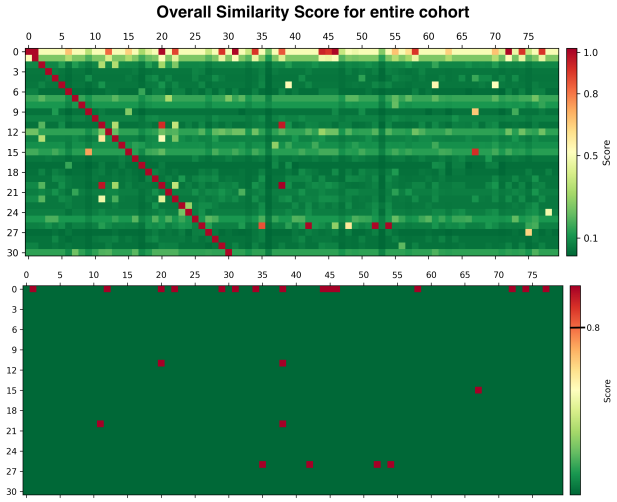

Fig. 1. Overall similarity score matrix: comparing 31 reports (y-axis) with 81 (x-axis). The top figure shows the overall similarity score and the bottom shows the matched cases, after removing self-similarity and applying threshold T_{AB}.

T_{AB} could be used to isolate the cases, where $\textbf{\textit{स}}_{A \leftarrow B}$ is high enough. For instance, setting a threshold of $T_{AB} = 0.8$ will identify all the cases where students have more than 80% of their work from a code component matched to another student's work.

Threshold T_{AB} allows us to use this tool in assessment from a variety of fields and topics. For some pieces of assessment, where the majority of code is expected to be similar, due to the nature of the tasks, where the same routine of code is used multiple times, T_{AB} could be chosen high. By contrast, if the assessment is designed in such a way, that very low similarity of code is expected, T_{AB} could be set to very low. Typically, the higher the value of T_{AB}, the less number of cases will be found, which are very similar.

E. Grouping the matched cases

After processing the entire cohort and computing the overall similarity score for each pair, we can create groups of students, who have similar work. Grouping students based on the similarity of work can further aid educators in reporting and processing misconduct cases effectively. Grouping of the students is done with a simplified approach, for example, if A matches with B, and B matches with C above the threshold, then A, B, and C belong to the same group.

F. Visualisation and validation

A software tool only providing the similarity score is not enough, as academics, we often have to analyses the details of misconduct; collusion, or plagiarism for reporting and validation. We implement the visualisation of a report, showing different blocks b_j from A matching to B. The visualisation also indicates the location of the identified blocks in code component C_c, which helps to locate the identified blocks in the original report. Similar visualisation can be shown for the plain-text matching. In addition to visualisation of the matched

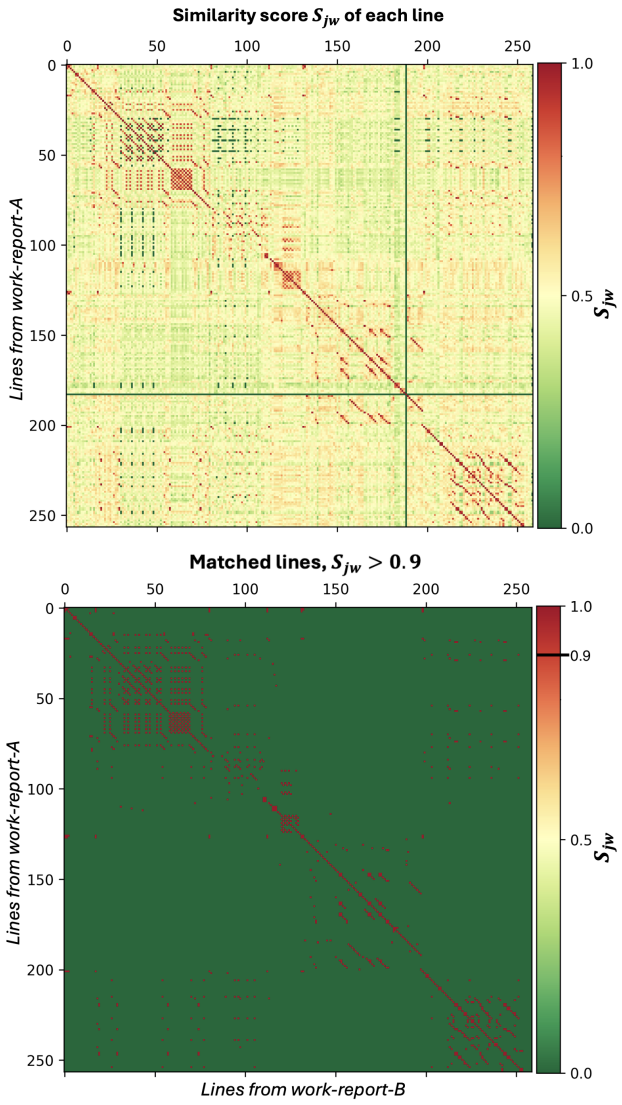

Fig. 2. Comparison of two reports: Top panel, similarity score of each line, and bottom panel, after applying threshold with $T = 0.9$, for report-26 and report-35).

block for a single comparison (with two reports), we also visualise the results of the entire cohort as a matrix.

G. Cloud-hosted Web Application for JBEval

This section outlines the development and deployment of our web application for JBEval designed to make JBEval accessible for educators through Web. The backend of the application is built using Python with FastAPI, and the front-end is developed with React and Next.js. AWS was selected as the cloud provider. The application follows a client-server architecture with the backend API served through FastAPI and the front-end rendered using Next.js.

This web-application allows users to upload two Jupyter Notebooks and visualise the full comparative analysis, or users can select two folders containing entire cohort(s) to produce an overall report.

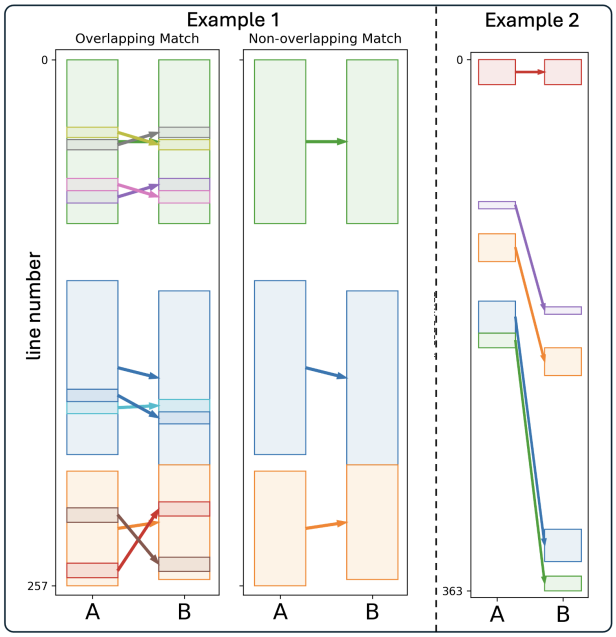

Fig. 3. Visualisation of block matching applied to two reports **A** and **B**, for computing similarity of A to B $\overline{\pi}_{A \leftarrow B}$. The vertical axis is the line number in Code Component C_c, and the placement of matched boxes indicates their position in the respective reports.

IV. Results and Discussion

In this section, we demonstrate the results obtained from a few selected example reports by using the proposed tool. The objective of this section is to provide further insights into results and visualisations.

A. Collection of assessment reports

For the purpose of demonstration, we chose a collection of reports, prepared in Jupyter Notebook and submitted as part of the assessment by our students. The assessment we chose was designed for the course on data analysis. The detailed instructions and guidelines for the tasks and questions for assessment were given in a well-structured text, written in Markdown, which renders as HTML elements in Jupyter Notebook.

The assessment guidelines also include a few segments of code (written in code cell) for initial set-up and to explain the tasks. The assessment included several tasks that students have to complete by coding in Python and included explanations in pre-allocated *raw* cell types.

We selected 30 reports (Jupyter Notebooks) submitted by students as part of their assessment and compared them with the same 30 reports along with additional 50 reports from the previous cohort. In total, we compare 30 reports with 80 reports. We also included an empty submission file named 0, as a reference.

B. Analysing the entire cohort

For comparing 31 reports (including reference file) across 81, we set $T = 0.9$, threshold for line matching, $L_m = 5$,

minimum number of lines for block matching, and for demonstration purposes, we use threshold $T_{AB} = 0.8$, i.e., if report A is 80% similar to B, it is considered as a match case (i.e. $\overline{s}_{A \leftarrow B} > T_{AB}$). The resulting matrix of overall similarity scores $\overline{s}_{A \leftarrow B}$ is shown in Figure 1. The bottom panel of Figure 1 shows very few match cases (11, 15, 20, and 26), as T_{AB} is high. It can be noticed, that report 0 has a high similarity with many reports, as it is an empty submission. However, it does not match 100% with all the reports, as students were required to change the part of the given code.

C. Comparison of two notebooks

Although, processing the entire cohort produces pairs of reports that are similar, the proposed tool allows the details comparison of two reports with visualisations. First, we show a line-wise comparison of a pair from one of the above-matched cases (report 26 and 35, with a score of 86%) as a matrix in Figure 2. Figure 2 indicates several lines matching in both reports, which is further processed by applying the block matching approach with $L_m = 5$.

The results of the block matching approach on the same pair (26 and 35) are shown as *Example 1* in Figure 3. *Example 1* in Figure 3 shows overlapping and non-overlapping blocks matching. The placement of blocks in figure corresponds to their position in the reports. The colors for the boxes are chosen at random to show distinct blocks. For instance, green box at the top in non-overlapping indicates that the first section of the report in 26 and 35 are the same, followed by a few lines of individual work. The overall similarity score for reports 26 and 35 is 86%, and Figure 3 helps to locate those matched blocks.

Example 2 in Figure 3 is selected to demonstrate the results for comparing two varied lengths of reports. In this case, two reports; namely; **2** and **11**, are chosen, where the code component of **2** has only 217 lines and **11** has 432 lines, almost twice. The overall similarity score $\overline{s}_{2 \leftarrow 11} = 0.34$ and $\overline{s}_{11 \leftarrow 2} = 0.19$. The above example shows the interpretation of the overall similarity score, which is asymmetric and the visualisation helps to validate the case where common blocks of code are located in both reports. In this particular example, report **2** had very minimal work done, and matched boxes are the provided code blocks in the assessment.

V. Conclusions

Coding-based assessments are being integrated in diverse field of courses and programs and Jupyter Notebooks provides a framework to create a single report that includes coding and description of the procedures and results. Detecting plagiarism and collusion largely rely assessors using manual approached. In this paper, we have introduced a tool, that is compatible with Jupyter Notebook and detects the similarity between students' reports. The algorithm implemented in this tool is parametrised allowing educators to tune the aggressiveness of similarity detection, which makes it suitable for assessments covering different learning outcomes. The current work is designed for Python, however, it can be adopted to Jupyter Notebooks that use different languages, such as R. We have also created an web application for proposed tool, with objective to release it in public domain that allows educators to access the system worldwide.

References

[1] M. Elhayany and C. Meinel, "Towards automated code assessment with openjupyter in moocs," in *Proceedings of the Tenth ACM Conference on Learning @ Scale*, ser. L@S '23. ACM, Jul. 2023, p. 321–325. [Online]. Available: http://dx.doi.org/10.1145/3573051.3596180

[2] H. Manzoor, A. Naik, C. A. Shaffer, C. North, and S. H. Edwards, "Auto-grading jupyter notebooks," in *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*, ser. SIGCSE '20. ACM, Feb. 2020, p. 1139–1144. [Online]. Available: http://dx.doi.org/10.1145/3328778.3366947

[3] J. T. Gulikers, T. J. Bastiaens, and P. A. Kirschner, "A five-dimensional framework for authentic assessment," *Educational technology research and development*, vol. 52, no. 3, pp. 67–86, 2004.

[4] A. Al-Gahmi, Y. Zhang, and H. Valle, "Jupyter in the classroom: An experience report," in *Proceedings of the 53rd ACM Technical Symposium on Computer Science Education-Volume 1*, 2022, pp. 425–431.

[5] J. W. Johnson, "Benefits and pitfalls of jupyter notebooks in the classroom," in *Proceedings of the 21st Annual Conference on Information Technology Education*, ser. SIGITE '20. ACM, Oct. 2020, p. 32–37. [Online]. Available: http://dx.doi.org/10.1145/3368308.3415397

[6] O. TOPSAKAL, "Teaching algorithms design approaches via interactive jupyter notebooks," *European Journal of Technic*, Jun. 2023. [Online]. Available: http://dx.doi.org/10.36222/ejt.1320404

[7] J. Wagemann, F. Fierli, S. Mantovani, S. Siemen, B. Seeger, and J. Bendix, "Five guiding principles to make jupyter notebooks fit for earth observation data education," *Remote Sensing*, vol. 14, no. 14, p. 3359, Jul. 2022. [Online]. Available: http://dx.doi.org/10.3390/rs14143359

[8] A. Al-Hashmi, A. Al-Abri, and K. Al-Riyami, "Investigating teachers and students' perceptions of academic plagiarism at the university level," *International Education Studies*, vol. 16, no. 6, p. 112, Nov. 2023. [Online]. Available: http://dx.doi.org/10.5539/ies.v16n6p112

[9] V. Beketov and M. Lebedeva, "Intellectual property and quality of education: Exploring the academic integrity among medical students," *Frontiers in Education*, vol. 7, Nov. 2022. [Online]. Available: http://dx.doi.org/10.3389/feduc.2022.1012535

[10] A. P. Koenzen, N. A. Ernst, and M.-A. D. Storey, "Code duplication and reuse in jupyter notebooks," in *2020 IEEE symposium on visual languages and human-centric computing (VL/HCC)*. IEEE, 2020, pp. 1–9.

[11] D. Gitchell and N. Tran, "Sim: a utility for detecting similarity in computer programs," *ACM SIGCSE Bulletin*, vol. 31, no. 1, p. 266–270, Mar. 1999. [Online]. Available: http://dx.doi.org/10.1145/384266.299783

[12] M. Joy and M. Luck, "Plagiarism in programming assignments," *IEEE Transactions on Education*, vol. 42, no. 2, p. 129–133, May 1999. [Online]. Available: http://dx.doi.org/10.1109/13.762946

[13] A. Aiken, "Moss: A system for detecting software plagiarism," *http://www. cs. berkeley. edu/˜ aiken/moss. html*, 2004.

[14] L. Prechelt, G. Malpohl, and M. Philippsen, "Jplag: Finding plagiarisms among a set of programs," 2000.

[15] 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, pp. 18–28, 2024.

[16] S. C. Cahyono, "Comparison of document similarity measurements in scientific writing using jaro-winkler distance method and paragraph vector method," *IOP Conference Series: Materials Science and Engineering*, vol. 662, no. 5, p. 052016, Nov. 2019. [Online]. Available: http://dx.doi.org/10.1088/1757-899X/662/5/052016