Education and Information Technologies
https://doi.org/10.1007/s10639-024-13218-5

GRAD-AI: An automated grading tool for code assessment and feedback in programming course

Ishaya Gambo[1] · Faith-Jane Abegunde[1] · Omobola Gambo[2] · Roseline Oluwaseun Ogundokun[3,4,5] · Akinbowale Natheniel Babatunde[6] · Cheng-Chi Lee[7,8]

Received: 23 August 2024 / Accepted: 26 November 2024
© The Author(s), under exclusive licence to Springer Science+Business Media, LLC, part of Springer Nature 2024

Abstract

The current educational system relies heavily on manual grading, posing challenges such as delayed feedback and grading inaccuracies. Automated grading tools (AGTs) offer solutions but come with limitations. To address this, "GRAD-AI" is introduced, an advanced AGT that combines automation with teacher involvement for precise grading, timely feedback, and personalized support, enhancing the education process. This paper aims to develop "GRAD-AI," an automated grading tool for computer programming assignments. The objective is to overcome the limitations of manual grading by harnessing AI's capabilities to deliver accurate and timely assessments, thus creating a more interactive and supportive learning environment. The results show that GRAD-AI provides unbiased grading and timely and accurate feedback delivery for programming assignments by using the Halstead Complexity Measure, Term Frequency – Inverse Document Frequency Measure, Abstract Syntax Tree Process, and K-means Clustering. GRAD-AI marks a substantial stride in improving grading and feedback delivery within the education sector. Its real-time feedback provision and gap identification contribute to enhanced learning outcomes. As AI's role expands, integrating automated grading tools like GRAD-AI becomes crucial for fostering personalized learning and adaptability. The paper underscores AI's potential to revolutionize assessment and grading processes, supporting global students' growth and development.

Keywords Automated grading tool · Feedback · Programming assignments · Assessment · Automated · Manual · Python programming · Artificial intelligence · Characteristic extraction · K-means clustering · Plagiarism detection · User Interface

Extended author information available on the last page of the article

Published online: 11 December 2024

1 Introduction

The rapid advancements in cloud computing, big data, machine learning, and artificial neural networks have paved the way for the development of machines that can mimic human intelligence. In this context, artificial intelligence (AI) refers to the capacity of machines to learn, recognize, react, observe, and solve problems (Lin et al., 2024; Wu et al., 2023; Zhai et al., 2021). The public often perceives AI as the ability of computers or machines to both think and act like humans. It encompasses the ongoing efforts to create automated systems replicating human thought and behavior (Jiang et al., 2024a, b; Li et al., 2024; Wartman & Combs, 2018). In essence, artificial intelligence entails the skillful imitation of human mental processes and behavior using software and tools (Mohammed & 'Nell'Watson, 2019).

While AI has made remarkable progress, there are fundamental distinctions between humans and AI in various aspects. These distinctions encompass emergence (AI is a product of human ingenuity, whereas humans possess innate reasoning and thinking abilities), nature (AI aims to replicate human behaviors and tasks, while humans exhibit adaptability through various cognitive processes), state (AI consists of digital machines, whereas the human mind is analogous), and function (AI relies on input data and predefined instructions, while humans leverage cognitive abilities, memory, and processing capacity), among others (Assaf et al., 2022; Fortuna & Gorbaniuk, 2022; Zhang et al., 2024). However, AI boasts several advantages over the human mind, including speed, connectivity, upgradability, and scalability (González-Calatayud et al., 2021).

The exploration of AI dates to Alan Turing's groundbreaking work in 1950, which introduced the concept of "thinking machines." Over the years, AI research has branched into diverse domains, resulting in a burgeoning body of literature (Liu et al., 2024a, b, c; Zhang & Aslan, 2021). Education is one such domain that has witnessed transformative changes due to emerging technologies. The integration of AI in education has gained significant attention in recent years (Zawacki-Richter et al., 2019). Notably, the 2018 Horizon study highlighted AI and blended learning technologies as significant advancements in educational technology, predicting a 48% increase in the use of AI in education between 2018 and 2022. Furthermore, AI applications in teaching and learning environments are expected to experience substantial growth (Chen et al., 2020; Liu et al., 2024a, b, c; Zhai et al., 2021).

As AI solutions advance, they play a pivotal role in identifying gaps in the education system, ultimately enhancing educational effectiveness (Chen et al., 2020, 2024; Katake & Sugandhi, 2024). AI contributes to personalization and efficiency, streamlining administrative functions to provide teachers/lecturers with more time and flexibility, which fosters adaptability—an inherent human capability (Pratama et al., 2023; Bahcelerli & Altınay, 2023; Kancan et al., 2023). The synergy between lectures, teachers, and machines can significantly improve student outcomes (Freeman et al., 2014; Haleem et al., 2022).

AI's application in education within the teaching and learning context encompasses three main areas: learning with AI (utilizing AI tools in education), learning about AI (educating individuals on the use of AI tools in education),

and preparing for AI (understanding the transformative potential of these tools) (González-Calatayud et al., 2021; Liu et al., 2024a, b, c; Xu et al., 2024a, b). AI in tutoring, for example, facilitates diverse teaching and learning approaches both within and beyond traditional classroom settings, offering round-the-clock support and granting access to learning resources for students in economically disadvantaged areas, irrespective of physical location (Chen et al., 2022; Jiang et al., 2024a, b; Xu et al., 2024a, b). AI in assessments involves conducting situational judgment tests through realistic student chatbot interactions. The AI system collects, scores, assesses responses, and provides feedback. It analyzes student responses to test questions and uses established algorithms to make predictions based on the data. This paper focuses primarily on AI in assessment.

Online assessment has been a research subject for the past two decades, particularly in providing prompt and accurate student feedback (Dawson et al., 2019; Gikandi et al., 2011; Jordan, 2012). However, the effectiveness of such assessments in providing satisfactory and accurate feedback to students remains a question. Hence, there is a need to develop a tool that can deliver relatively accurate assessments that are also satisfactory to students. Various fields have extensively employed automated grading tools to assess and grade student assignments. These tools enable teachers/lecturers to automatically evaluate student solutions against predefined rubrics and offer immediate feedback on the accuracy of their work, empowering students to track their progress and make improvements (Heckman & King, 2018). Unfortunately, automated grading tools are underutilized in many developing countries, including Nigeria. In these regions, students often rely on traditional, manual methods for completing assignments, using pen and paper, rather than embracing the technological shift toward online assignments and automated grading tools, which require access to digital devices. This discrepancy arises from the perception that education systems in developed countries benefit from superior assessment and grading practices. This advantage is attributable primarily to the adoption of online assignment submission and automated grading tools (Petropoulou et al., 2014).

This situation presents a significant challenge: the absence of instant feedback on student assignments. Students' assignments sometimes remain unassessed or ungraded, depriving them of insights into areas needing improvement. As the number of students continues to grow, along with the effort required to grade their assignments, there is a pressing need for automated grading tools to provide timely and accurate feedback. Furthermore, as society increasingly embraces technology, the future of education will likely involve the widespread use of automated grading tools. Technology permeates all fields of study and endeavors, including submitting student assignments. For instance, one way to address the issue of assessing and grading programming assignments is to instruct students to test their code solutions and then evaluate them based on task requirements, checking for code correctness (Edwards et al., 2014). This approach empowers students to identify areas for improvement in their coursework.

The broad utility of automated grading tools lies in their ability to assess, grade, and deliver immediate, precise feedback, surpassing the capabilities of human assessors. This paper aims to study, observe, and analyze existing applications designed for assessing and grading assignments while providing feedback. Subsequently, it seeks to

introduce a superior solution called "GRAD-AI," enabling students to complete assignments online, receive assessments and grades, and access accurate feedback.

The implemented tool specifications focused on a computer programming course, "Introduction to Python Programming," which solely covers the introductory aspects of Python programming as outlined in CSC201 and CSC202/208 manuals, which forms part of the curriculum at the Department of Computer Science and Engineering, Obafemi Awolowo University, Ile-Ife, Nigeria. The contribution of this paper is fivefold. Theoretically, GRAD-AI contributes knowledge by advancing the understanding of automated grading systems in programming education. It explores theoretical concepts from artificial intelligence and machine learning to enhance the tool's ability to analyze and evaluate programming assignments effectively. Systematically, GRAD-AI development presents a well-structured and comprehensive approach to automated grading in programming courses. The tool systematically integrates syntax and semantic error checks, personalized feedback, and lecturer review, ensuring a cohesive and efficient grading system. Methodologically, GRAD-AI introduces innovative techniques for code assessment and feedback. It detects semantic differences by leveraging the Halstead complexity measure, term frequency-inverse document frequency measure, and abstract tree process. It predicts ways to fix syntax errors, providing valuable insights into students' coding practices. Practically, GRAD-AI addresses real-world challenges faced by students and teachers/lecturers in programming courses by implementing an automated grading tool that aids the grading process and returns feedback. Automating the grading process saves lecturers time and gives students instant feedback for improved learning outcomes. Analytically, GRAD-AI critically evaluates and validates the effectiveness of the automated grading tool. Through rigorous testing and comparison, it provides analytical insights into the tool's performance, contributing to the broader understanding of automated grading systems' strengths and limitations.

The rest of the paper is structured as follows. Section 2 delves into the motivation and problem statement, providing a foundation for the study. In Sect. 3, the paper presents a comprehensive review of related works, identifying areas that demand improvement. Section 4 outlines the methodological design approaches, elucidating the implementation's requirements and specifications. The subsequent section, Sect. 5, details the actual system implementation, validating the design specifications' accuracy. Section 6 delves into the procedures employed for system testing and presents the results and discussion. Section 7 addresses the limitations of the study. Section 8 presents the threats to validity, while Sect. 9 offers insights into potential future work. Concluding remarks drawn from the research are encapsulated in Sect. 10, bringing the paper to a thoughtful conclusion.

2 Motivation and problem statement

The current educational system heavily relies on manual grading to evaluate students' performance, resulting in significant challenges, particularly with the increasing student-to-teacher ratio. Manual grading often leads to delayed or absent feedback for students, potentially causing future assessment failures due to a lack of awareness of their weak areas. Moreover, manual grading typically focuses solely on

grades without addressing specific areas of improvement. One of the primary issues with manual grading is its inconsistency, as it may not consistently assess and grade student assignments accurately, resulting in unreliable feedback. Additionally, the manual grading approach cannot analyze individual learning paces and needs due to its non-interactive nature, unlike automated grading tools (AGTs).

Concurrently, researchers and programmers have actively explored the development of automated grading tools that evaluate students' assignments, including programming tasks. However, while some of these automated grading tools offer distinct benefits, they also exhibit certain limitations. Some tools solely provide grades without offering detailed feedback to students, while others may provide feedback but lack a grading component. Specific automated grading tools require the execution of programming assignment codes for assessment, rendering them incapable of evaluation if the code cannot be executed, even if it contains correct syntax and semantics. Furthermore, some automated grading tools lack the crucial feature of allowing teachers to review students' assignments and add personalized comments, limiting the overall effectiveness of the grading process and hindering the provision of tailored guidance to students.

Therefore, this paper's primary objective is to develop an advanced automated grading system called "GRAD-AI" that not only addresses the limitations of the existing manual assessment approach but also overcomes the shortcomings of some current automated grading tools. The proposed automated grading tool will provide timely and accurate feedback, analyze individual learning needs, and foster an interactive environment to support students' learning journey. By incorporating features for both precise grading and detailed feedback, including the ability for teachers to review assignments and add personalized comments, the new automated grading tool aims to enhance the overall efficiency and effectiveness of the grading process. This comprehensive and well-rounded approach to automated grading will promote teacher-student interaction, ultimately creating a more robust and supportive educational ecosystem that leverages the advantages of automation while upholding the value of teacher involvement and personalized learning experiences. The following research questions have been addressed in this paper.

- RQ1: Can GRAD-AI address the challenges of manual grading and provide timely and accurate feedback to students?
- RQ2: Can GRAD-AI allow teachers to review students' assignments and add personalized comments?
- RQ3: Can GRAD-AI predict ways to fix syntax errors in student code submissions, contributing to a more comprehensive feedback mechanism?
- RQ4: How does GRAD-AI contribute to the ongoing efforts of teachers/lecturers in developing nations to transition from traditional teaching methods to technologically advanced learning environments?

The response to RQ1 is explained in Sect. 6, where the implemented system undergoes testing to assess its capability to provide timely and accurate feedback. For RQ2, Sect. 4 addresses the inquiry by delineating the methodological design approach, presenting a comprehensive overview of the design for all aspects or features of the tool to be implemented. To address RQ3, pertinent information can be

located in Sect. 5. This section extensively discusses the actual system implementation, highlighting K-Means Clustering as one of the methods employed to address syntax errors in codes. The examination of how GRAD-AI contributes to the ongoing efforts of teachers/lecturers in developing nations, as posed in RQ4, is thoroughly discussed in Sects. 2, 4, 5, and 6.

3 Related works

Recent advancements in Automated Grading Tools (AGTs) have significantly broadened the scope of academic assessment, transcending the traditional confines of programming assignments. Pioneering research by Bian et al. (2020) and Stikkolorum et al. (2019) has been instrumental in the automated evaluation of class diagrams. Concurrently, Alur et al. (2013) have substantially contributed to developing AGTs for Deterministic Finite Automaton (DFA) constructions. Bhangdiya et al. (2015) have been at the forefront of automating SQL query assignment grading in database education. The field of essay grading has also witnessed a paradigm shift with seminal works by Valenti et al. (2003), Hearst (2000), Ramalingam et al. (2018), and Ramesh and Sanampudi (2022), highlighting the versatility and potential of AGTs in evaluating diverse forms of student work.

However, a few works have surfaced within the specific niche of AGTs tailored for programming assignments, albeit with notable limitations. While these AGTs have made strides in automating the grading process for coding tasks, they often fail to address key aspects crucial for a comprehensive and practical assessment. Grad-AI emerges as a promising solution, aiming to overcome these limitations and provide a robust framework for the automated grading of programming assignments.

Within the sphere of AGTs, Parihar et al. (2017) 's contributions are noteworthy, particularly with their development of 'GradeIt.' This tool has distinguished itself through its proficiency in grading and syntax verification, addressing the specific requirements of programming assignments. Despite its strengths, 'GradeIt' demonstrates notable limitations, such as the absence of features designed for educators' convenience and oversight of semantic error analysis in the evaluated code.

Shifting the focus to the work of Heckman and King (2018) and Heckman et al. (2020), their integration of Github and Jenkins offered an innovative approach to automated grading. This dynamic duo of tools facilitated automated grading and collaboration, covering various stages of software engineering. Despite its commendable features, this AGT faltered by overlooking semantic and syntax errors, crucial components for a comprehensive evaluation. Additionally, the absence of a dedicated lecturer review feature further limited its potential.

In the evolving landscape of AGTs, the 'Auto-grader' developed by Liu et al. (2019) has emerged as a specialized tool that prioritizes semantic discrepancies in programming tasks. This innovative approach introduces an additional layer of complexity to the grading process, enhancing the depth of assessment. Despite its novel focus, 'Auto-grader' presents significant limitations, including the omission of syntax error detection and the lack of an integrated grading mechanism, which may limit its practicality in academic environments.

Galassi and Vittorini (2021) further advanced the AGT landscape with their tool, introducing debugging capabilities and supervised classification for unlimited answers. While this tool elevated AGTs by providing valuable feedback and reviews, it fell short by not predicting error fixes and lacking both syntax and semantic error checks and a dedicated grading feature.

Nabil et al.'s (2021) introduction of 'EvalSeer' into the AGTs landscape marks a significant step forward with its cloud-based learning management system capabilities. 'EvalSeer' excels in detecting syntax errors and offering corrective predictions, streamlining the grading process for programming assignments. Despite these advancements, 'EvalSeer' encounters challenges in conducting semantic error analysis, lacks a bespoke grading function, and omits the crucial element of lecturer feedback, which is essential for a holistic assessment framework.

In summary, these AGTs collectively contribute to improving the teaching–learning environment, yet they face inflexible grading systems, lack of semantic error checks, and insufficient feedback. The proposed Grad-AI strives to build upon these foundations by addressing these limitations, incorporating both syntax and semantic error checks, offering a flexible grading feature, and providing comprehensive feedback. Grad-AI endeavors to revolutionize the landscape of automated grading tools for programming assignments through these enhancements.

4 Methodology

This section focuses on the development of GRAD-AI, an automated grading tool. The chosen model and system requirements, as outlined in Sirhaug (2018), form the foundation of the tool. To systematically plan and implement the tool, we followed a well-defined development model with crucial steps, ensuring progress stays on track. GRAD-AI directly addresses the issues identified in Sect. 2, providing practical solutions. As per Tarcsay (2023), the data collection method was adopted, involving the gathering and analysis of coding questions and answers in the CSC202/208 – Computer Programming course. The GRAD-AI model encompasses various stages, including data analysis, design specifications, implementation, training, and testing. Figure 1 illustrates the AGT approach's context diagram, depicting a seamless flow from student registration to logging in and assignment submission on the dashboard created to the assessment of the submission, and then grading and feedback, which are returned to the dashboard. The lecturer/ teacher can also add reviews after checking the students' assignment solution, which influences the grades and feedback returned.

The system requirement specifications play a pivotal role in facilitating interaction between students and lecturers regarding the semantics and syntax of Python programming codes. Prior to specific design stages, a thorough assessment of needs is conducted to minimize the need for redesign. The system requirements include enabling teachers to add reviews, accessing students' assessments and details, and assigning grades. Likewise, students should be able to access their grades, receive feedback, and input personal details and assignment solutions. Admin functionalities involve accessing students' details and authorizing teachers' login credentials,

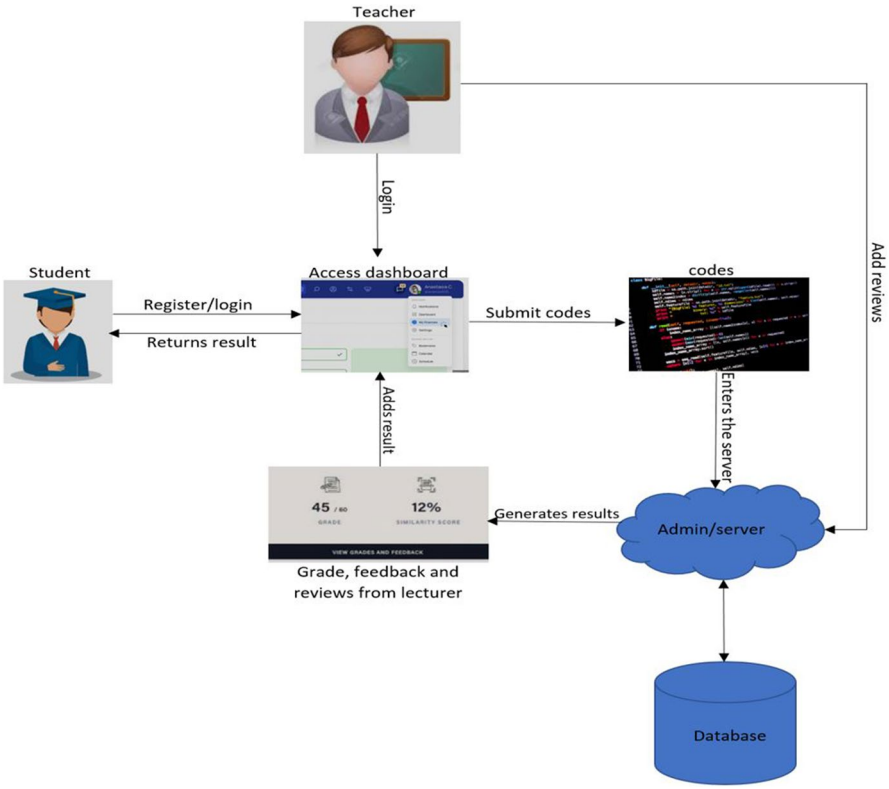


Fig. 1 Context diagram

ensuring a comprehensive and user-friendly experience for all stakeholders in the automated grading process.

4.1 Data collection and analysis

The data employed in this study originated from the practical class exercises of CSC202/208, "Introduction to Python Programming II," conducted in the Department of Computer Science and Engineering, Obafemi Awolowo University, Ile-Ife, Nigeria, in 2024. This dataset in Table 1 encompassed submissions from 50 students, comprising responses to five questions sections. These submissions were sourced from a dedicated platform for students to submit solutions to their assignments. Consequently, variations in responses emerged due to the diverse approaches taken by students. Each assignment was graded based on the accuracy of the student's code and its alignment with expected outputs. Additionally, the system captured subtask scores, forming a detailed 250-record dataset used in model training and evaluations. Clustering for Grading: One core algorithm in GRAD-AI's grading system is K-means clustering, which helps categorize student submissions based on performance. Data points representing submission scores

Table 1 Total number of students with attained grades per question

Question No	1	2	3	4	5
No of people who had 10/10	4	9	2	6	5
No of people who had 9/10	6	3	7	4	4
No of people who had 8/10	8	4	5	3	6
No of people who had 7/10	5	6	6	8	7
No of people who had 6/10	7	5	4	5	3
No of people who had 5/10	3	8	8	7	8
No of people who had 4/10	6	5	3	2	5
No of people who had 3/10	5	5	5	6	6
No of people who had 2/10	3	4	6	5	4
No of people who had 1/10	3	1	4	4	2

are grouped into clusters to ensure consistent and fair grading. The closest cluster center is identified for each data point, and each student's performance is assigned to a corresponding grade range within a cluster. This method minimizes the subjective aspects of grading by creating standardized performance categories.

The steps involved in K-means clustering are as follows:

Let S = {s1, s2,…,sn} represent the set of data points and A = { a1, a2,…,an} represent the set of centres;

1. Select "x" cluster centers randomly
2. Get each cluster center and data point distances
3. Cluster center with the minimum distance should be assigned the data point
4. Get a new cluster center using Eq. 3.5:

$$a_{n=}\left(\frac{1}{x_n}\right)\sum_{m=1}^{x_n} s_n \tag{1}$$

Where "xn" is the number of data points in an nth cluster in Eq. 1

5. Repeat steps 2
6. Repeat step 3 if a data point is reassigned; otherwise, stop

In the cluster table in Table 2, the numbers in each cell represent the cluster assignment for the corresponding combination of question and score. These assignments were determined by minimizing distances between data points and cluster centroids, following the iterative steps of the K-means algorithm.

4.2 Design specification

In this stage, the data was thoroughly examined and translated into visual designs illustrating the users' anticipated experience. The analysis aimed to guarantee that the users' requirements and expectations were encompassed and mirrored

Table 2 Cluster table after K-means clustering was applied Assume K = 3

Score	Cluster 1	Cluster 2	Cluster 3
10/10	1	2	3
9/10	2	1	3
8/10	3	1	2
7/10	1	2	3
6/10	3	2	1
5/10	2	1	3
4/10	3	1	2
3/10	1	2	3
2/10	3	2	1
1/10	3	1	2

Table 3 Functional Requirements

S/N	SPECIFICATIONS
1	Teachers can add reviews for students' assessments
2	Teachers can access students' assessment data
3	Teachers can access students' personal details
4	Teachers can input and manage grades for assignments
5	Students can access their grades
6	Students receive timely and constructive feedback
7	Students can input personal details (name, reg. number, etc.)
8	Students can submit assignment solutions for assessment
9	Admins can access detailed information about students
10	Admins can authorize teachers' login details

throughout each phase of the design process. The design was realized using case, class, sequence, and block diagrams, delineating user connections and system operations. Table 3 presents the functional requirements of the system.

The architecture depicted in Fig. 2 comprises six significant entities. These entities include the grading components responsible for evaluating submitted assignments, resources such as downloadable files or assignments for students, system configuration for determining the overall assignment grade and displaying result details, a dashboard enabling students to review feedback and grades, a compiler for code compilation, execution for running compiled code, and results returned after execution. Figure 2 illustrates the representation and interaction among these entities.

a) ***Grading Module***: The grading module evaluates each student's code based on syntax correctness, structure, and adherence to best practices. The module assesses readability and efficiency by combining abstract syntax tree (AST) parsing and complexity measures (e.g., Halstead complexity). This analysis enhances

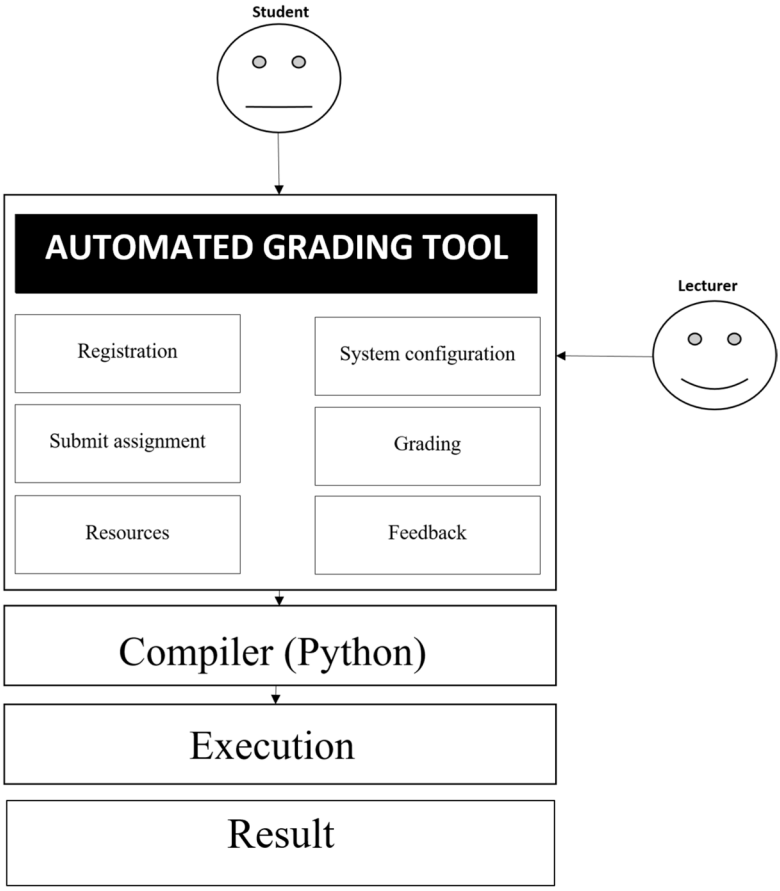


Fig. 2 An overview of the architecture of the automated grading tool

grading accuracy by factoring in code structure and quality rather than just functionality.

b) ***Dashboard for Teachers***: The teacher's dashboard allows educators to view and assess students' assignments in a centralized location. Teachers can also add specific feedback for each student, contributing to a personalized learning experience. The dashboard further supports a tabular display of each student's grades, fostering an efficient grading workflow.

c) ***Student Dashboard***: Students interact with GRAD-AI primarily through their personalized dashboard. This interface provides access to grades and feedback, allowing them to understand performance areas and receive guidance on improving their skills. The dashboard encourages active participation and engagement by including interactive components like randomly generated questions.

d) ***Compiler and Execution Module***: After students submit their code, it is compiled and executed to validate outputs against expected results. This module uses sand-

boxed environments to ensure secure execution, minimizing risks from potentially erroneous code. The results are then used as input for the grading module.

Use case analysis, as demonstrated in Fig. 3, clarifies the actions performed by actors within the system to define its functionality. This analysis expresses the system's context and is valuable for representing scenarios in which the system interacts with organizations, individuals, or external systems. Figure 3 displays a Unified Modeling Language (UML) Use Case diagram outlining the Automated Grading Tool (AGT) functionalities.

The class diagram illustrates the system's static structure, as seen in Fig. 4. This diagram outlines classes, attributes, operations, and relationships between objects. Figure 4 showcases the superclass "user," comprising the admin and lecturers, and the child class "user," representing students. Associations include a 1-to-many relationship between the admin class and assignment, lecturer, profile, and user classes, as well as many-to-many relationships between the lecturer class and assignment and profile classes, and similarly for the student class.

The sequence diagram in Fig. 5 shows time-ordered interactions and operation executions between objects. This diagram details the steps involved in scenarios related to use case realization. In Fig. 5, the student registers, logs in, views their profile and submits an assignment to the admin, who manages user data and verifies the lecturer. The lecturer configures the system, assesses the assignment, and adds reviews or comments, which are then returned to the admin. The admin assesses the assignment and provides grades, feedback, and lecturer comments to the student.

Figure 6 illustrates the relationships between essential functionalities, represented by blocks, with connected lines denoting their relationships.

5 System implementation

The implementation of GRAD-AI focuses on creating an automated grading system that provides objective scoring, feedback, and analysis of code assignments. The GRAD-AI scoring mechanism is central to this, as it integrates multiple algorithms to evaluate code quality, readability, and adherence to assignment requirements.

5.1 Scoring mechanism

The GRAD-AI scoring mechanism is designed to assess programming assignments comprehensively by evaluating various aspects of students' code submissions. This mechanism incorporates several advanced techniques to ensure an accurate, fair, and multifaceted assessment, making it possible to provide detailed feedback on code quality, complexity, logical consistency, and conceptual understanding.

The scoring process comprises four main components:

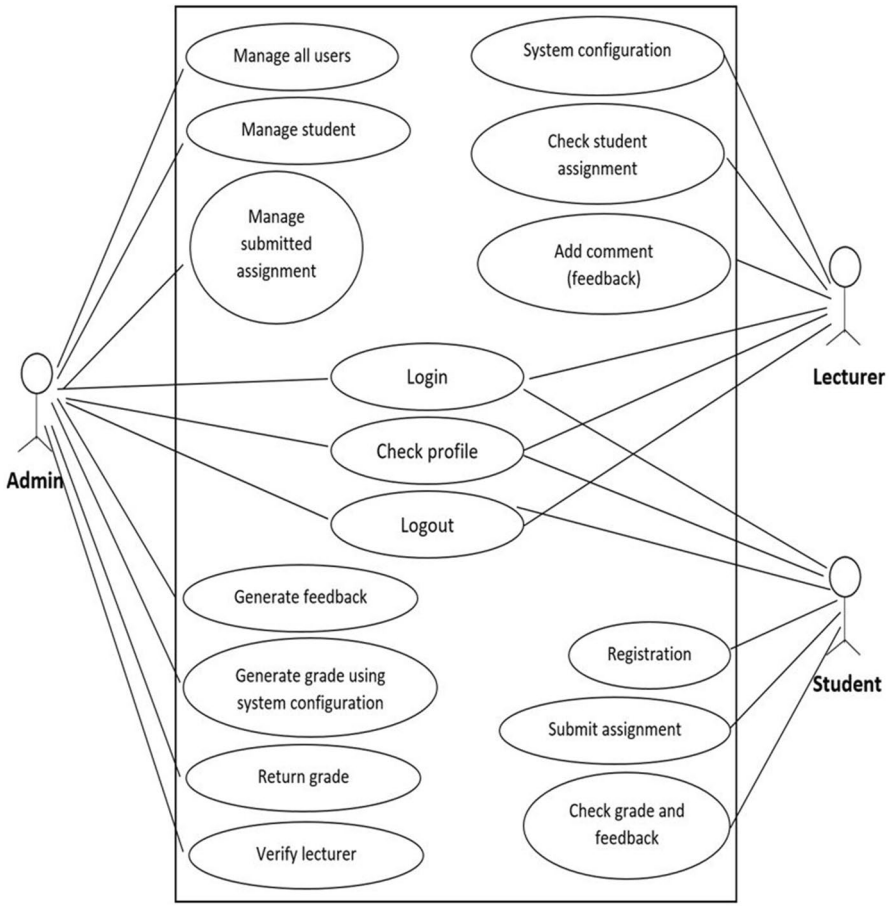


Fig. 3 Use case diagram

1. **Halstead Complexity Measures**: Halstead metrics are essential for analyzing code complexity based on the structure and content of the code. Key metrics include:

   a) **Program Length (N)**: The code's total number of operators and operands. This measures the extent of the code and gives insight into its length and overall complexity.
   b) **Vocabulary (n)**: The count of unique operators and operands. A high vocabulary count often indicates more diverse operations and variables, contributing to code intricacy.
   c) **Volume (V)**: Defined as $V = N \times \log_2(n) V = N \times \log_2(n)$, this metric quantifies the amount of information or cognitive effort required to understand the code.

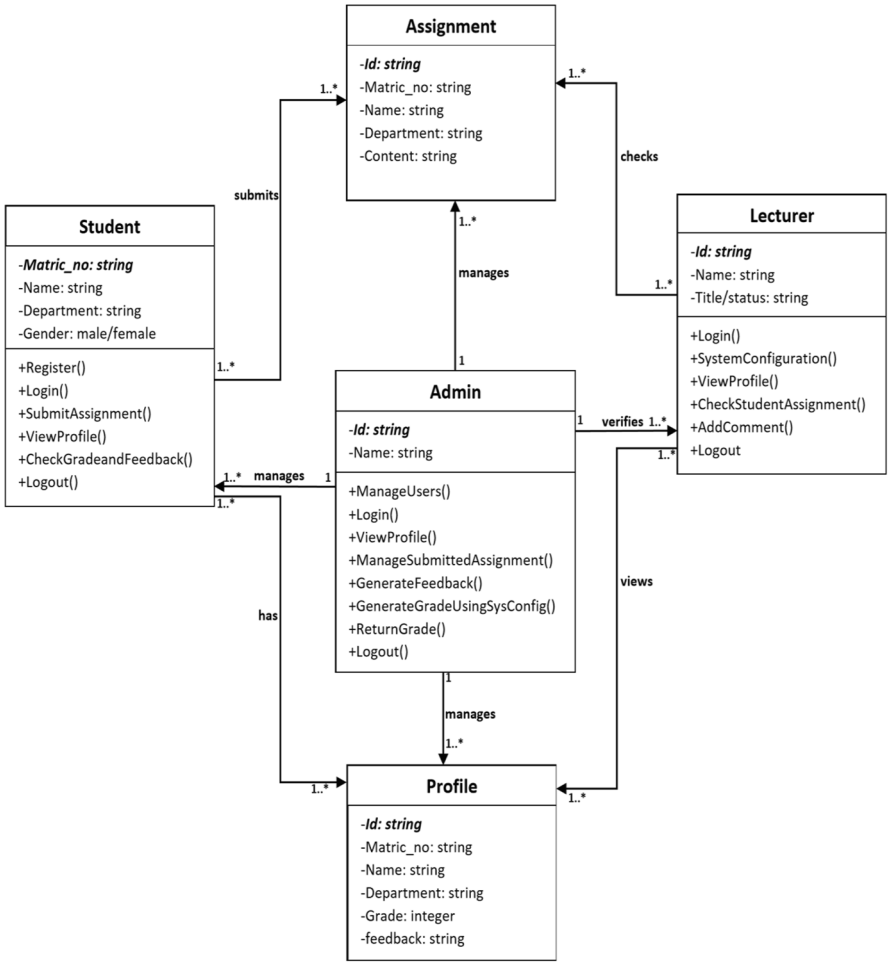


Fig. 4 Class diagram

d) **Effort (E)**: Represented by $E = V \times DE = V \times DE = V \times D$, where DDD is a difficulty, this measure approximates the effort needed to comprehend and implement the code.

Analyzing these metrics, GRAD-AI can evaluate how readable, manageable, and concise a student's code is. For example, excessive program length or high effort scores might indicate overly complex code, prompting feedback on simplifying structures or improving readability.

2. **Abstract Syntax Trees (AST)**: ASTs are used to analyze the syntactic structure of the code. Unlike mere syntax checks, ASTs decompose code into its fundamental components (e.g., statements, loops, and expressions) and represent it in a

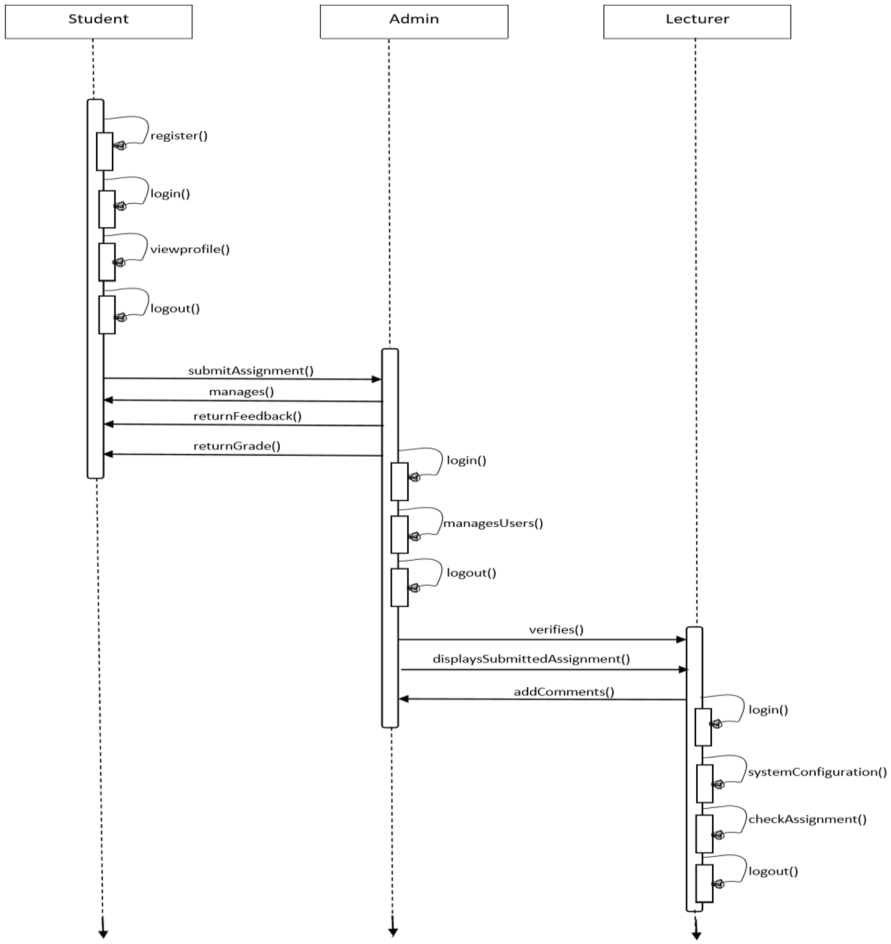


Fig. 5 Sequence diagram

tree structure, capturing hierarchical relationships and dependencies. This allows GRAD-AI to:

a) Detect structural or logical errors in code flow.
b) Ensure that code follows logical sequences (e.g., initialization before use of variables).
c) Validate the correctness of control structures and nested statements.

GRAD-AI can spot errors that might otherwise go undetected in traditional syntax checks, such as misplaced conditions or inconsistent variable usage, and to provide targeted feedback on improving logical structure by parsing the code using ASTs

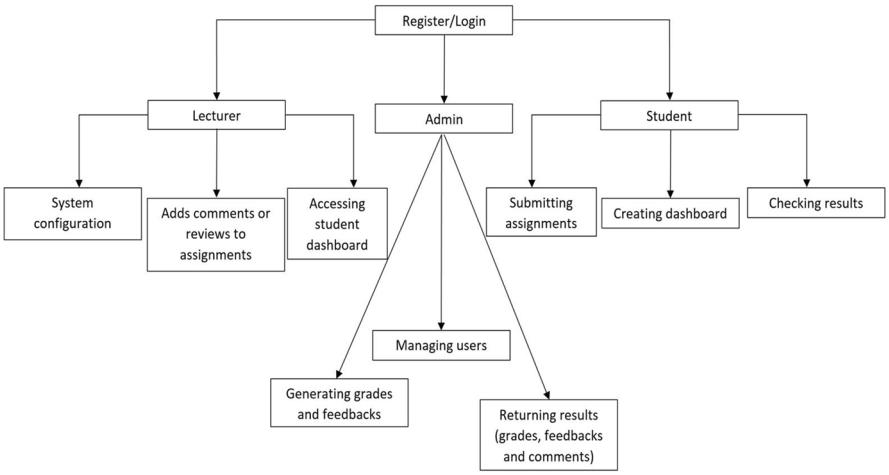


Fig. 6 Block diagram

3. **Term Frequency-Inverse Document Frequency (TF-IDF)**: TF-IDF is a statistical measure typically used in text analysis but applied here to evaluate the comments and function descriptions within students' code. The mechanism works by:

   a) Calculating term frequency, which represents how often specific programming terms or keywords (e.g., "loop," "function," "variable") appear in comments.
   b) Adjusting by inverse document frequency reduces the weight of standard terms that are less informative across different submissions.

This component helps identify whether students understand and appropriately describe their code functions. For example, if a student's comments repeatedly misidentify constructs or omit key terms, GRAD-AI can provide guidance on best commenting practices, terminology use, and clarity in explanations.

4. **K-Means Clustering**: K-means clustering groups students' submissions into clusters based on performance metrics such as complexity, correctness, and consistency. Each submission is evaluated for these metrics and assigned to a cluster, with clusters indicating various performance levels:

   a) **High-Performing Cluster**: Accurate and efficient code, demonstrating strong comprehension and adherence to programming principles.
   b) **Mid-Level Cluster**: Code that meets basic requirements but may need improvement in structure or readability.
   c) **Low-Performing Cluster**: Code with significant errors or deviates significantly from expected solutions.

Clustering allows GRAD-AI to deliver individualized and consistent feedback across submissions in each performance category. For instance, all submissions within a low-performing cluster may receive feedback on foundational improvements, while mid-level submissions could receive tips on optimizing code efficiency.

Together, the four components listed above ensure that the GRAD-AI scoring mechanism not only evaluates technical correctness but also encourages good programming practices, clarity, and readability, all of which are vital for student development in coding.

5.2 System modules

The GRAD-AI system architecture includes several core modules, each contributing to different stages of the grading, feedback, and user experience processes. These modules ensure a seamless workflow from code submission to grading and feedback generation, effectively supporting students and educators.

1. **Grading Module**: The Grading Module is the central component for analyzing code submissions. Using the AST and Halstead Complexity metrics, this module assesses:

    a) **Syntax and Semantics**: ASTs enable the module to break down the code into syntactic units, ensuring that syntax and logical structures adhere to programming standards.
    b) **Readability and Complexity**: By applying Halstead metrics, the module quantifies complexity and readability. High complexity scores trigger feedback on potential simplification.
    c) **Functional Correctness**: To verify functional correctness, the module compares the code's output with expected results. If the output aligns, the code passes the functional correctness check; otherwise, targeted suggestions are given on code logic.

The Grading Module synthesizes these assessments to generate a final score, factoring in whether the code works and how it is structured, making for a holistic evaluation process.

2. **Feedback Generation**: This module transforms grading insights into actionable feedback. Using TF-IDF and K-means clustering, the system identifies common patterns or recurring issues within each submission cluster and provides relevant feedback. Key functionalities include:

    a) **Targeted Comments**: Feedback for low-performing clusters may include foundational advice, while mid-level clusters receive tips on code optimization.

b) **Improvement Suggestions**: The feedback module generates specific guidance based on TF-IDF analysis, pointing out unclear or incomplete comments and suggesting terminology corrections or enhancements.
c) **Error Correction Tips**: For syntax errors detected via AST analysis, the feedback includes possible fixes, such as correcting mismatched braces or uninitialized variables. The system provides explanations and examples for logical errors to help students resolve issues independently.

This detailed, customized feedback not only addresses issues in the code but also helps students understand why these issues matter, reinforcing fundamental programming principles.

3. **Dashboard Interface**: The Dashboard Interface serves as the main point of interaction for students and teachers. Key features include:

   a) **Student Dashboard**: Students access grades, feedback, and detailed performance analysis. The interface presents data in an organized, visually accessible manner, enabling students to review feedback, track progress, and understand areas for improvement.
   b) **Teacher Dashboard**: Teachers can view an aggregated analysis of student performance, making it easier to identify overall trends or common issues. They can also access individual submissions for further review, add personalized comments, and adjust scores if necessary.
   c) **Interactive Elements**: The dashboard incorporates interactive features such as a Q&A section, which allows students to ask follow-up questions, and a feedback tab, where students can rate their experience or request clarification on feedback.

4. **Compiler and Execution Module**: After students submit their assignments, this module compiles and executes the code in a secure, sandboxed environment. It ensures:

   a) **Output Verification**: Code output is tested against expected results to verify accuracy.
   b) **Security**: Using sandboxing, the system avoids potential security risks from untrusted code.
   c) **Real-Time Execution Results**: Execution feedback is returned instantly, allowing for immediate error identification.

This module is beneficial for assignments requiring hands-on testing because it verifies code functionality directly within the grading system, reducing manual workload.

5. **Error Handling and Notification**: This module manages system errors, guiding users through troubleshooting steps for failed submissions or syntax errors. It also includes:

a) **Syntax and Semantic Error Identification**: For each submission, the system flags common issues such as unmatched parentheses, undeclared variables, or misused operators.
b) **Correction Suggestions**: The error-handling module flags errors and suggests corrections. For example, if a student leaves a variable uninitialized, the system prompts them to add a value.
c) **Submission Notifications**: When students submit assignments, the system confirms them and highlights any detected issues before the final submission. This prevents errors from affecting grading and allows students to revise their code.

Each of these modules plays a critical role in ensuring GRAD-AI's functionality is a reliable and user-friendly automated grading tool. Altogether, they create an integrated system that supports student learning, provides detailed and constructive feedback, and reduces grading time for educators, helping foster a more interactive and effective educational environment.

Table 4 provides an overview of the development tools utilized in implementing the AI-based grading system, covering diverse functionalities. Visual Studio Code functions as a versatile Integrated Development Environment (IDE) supporting multiple programming languages, Git and GitHub for version control and collaboration, TensorFlow and PyTorch as open-source AI frameworks, Flask for web development, HTML and CSS for frontend design, Firebase for cloud services, and Python's unittest or pytest for testing. These tools collectively facilitate various aspects of the grading system's development, including coding, version control, AI modeling, frontend design, cloud integration, and testing using a Google Form for functional testing.

Moving to the system requirements outlined in Table 5, these are essential specifications and capabilities that the Automated Grading Tool (AGT) must meet to fulfill its intended functions effectively. These requirements include sufficient hardware, compatible software, effective AI models and algorithms, an intuitive user interface, and scalability. Ensuring powerful hardware, compatibility with major operating systems, utilization of AI frameworks, well-trained models, Natural Language Processing (NLP) capabilities, user-friendly design, and efficient performance is crucial for achieving accurate grading and user satisfaction within the AGT.

The development process spanned approximately four months, with an additional two months allocated for the pilot implementation. Throughout this period, the authors prioritized the creation of a user-friendly interface accessible to both students and teachers/lecturers. Figures 7, 8, 9, 10 and 11 illustrate various functionalities of the system:

- The Teacher and Admin Login Page is shown in Fig. 7. It serves as the first point of access for teachers and administrators. Its main objective is to get users accessing the grading system to provide their usernames and passwords. This interface uses a robust server-side authentication method and acts as a gateway. The system is intended to offer unambiguous and easily navigable error messages if login failures arise, enhancing the overall usability of the login proce-

Table 4 Essential development tools employed at different stages of the AGT

S/N	Type of Development Tool	Tool Used	Description
1	Integrated Development Environment (IDE)	Visual Studio Code	A versatile IDE with extensive plugin support for various programming languages (python) and frameworks
2	Version Control System	Git	Git is a popular version control technology, and the GitHub platform was used to host and collaborate on remote repositories
3	AI Frameworks and Libraries	TensorFlow	A Google machine learning and deep learning framework that is open-source
		PyTorch	a well-known deep learning framework created by the AI Research team at Facebook
		scikit-learn and spaCy	scikit-learn and spaCy were utilized for tasks such as natural language processing (NLP)
4	Web Framework (if applicable)	Flask	A lightweight Python web framework suitable for small to medium sized projects
5	Front end development	HTML (Hypertext Markup Language)	It is used to structure the content of the web. The elements, such as headings, paragraphs, buttons, forms, etc., and their locations on the page are described. HTML provides the backbone of the user interface by creating the structure and layout of the grading tool
		CSS (Cascading Style Sheets)	It specifies the layout and look of HTML components. It selects the user interface's colours, fonts, spacing, and other visual elements. CSS ensures that the grading tool has a consistent and aesthetically pleasing design
6	Cloud Services	Fire base	It is a Google-backend platform for building mobile and web applications. It offers cloud-based services like real-time database, authentication, cloud messaging and hosting
7	Testing Framework	Python's built-in unittest or libraries like pytest	• A testing framework is essential for ensuring the reliability and correctness of the software. Python's built-in unit test or libraries like pytest were used for testing purposes
		Google form	A Google form was sent for functional testing to be filled out by users

Table 5 The key system requirements for GRAD-AI

S/N	Type of System Requirement	System Requirements
1	Hardware Requirements	a. Sufficient computational power and memory to handle AI algorithms and processing b. Adequate storage capacity to store grading data, feedback, and user information c. Fast and reliable internet connectivity
2	Software Requirements:	a. Operating System: Compatible with major operating systems like Windows, macOS, and Linux b. Programming Languages: Suitable programming languages for AI development, i.e. Python c. AI Frameworks: Utilize AI frameworks like TensorFlow, PyTorch, or scikit-learn for machine learning tasks
3	AI Model and Algorithm Requirements	a. A well-trained AI model capable of accurately assessing and grading assignments b. Machine learning algorithms are used to process and analyze student submissions c. Natural Language Processing (NLP) capabilities to interpret textual responses effectively
4	User Interface and Experience (UI/UX) Requirements	a. Intuitive and user-friendly interface for students to submit assignments and view grades/feedback b. Clear and straightforward instructions are provided so users can understand the tool's functionalities c. Responsive design to ensure accessibility across different devices (e.g., desktops, tablets, smartphones)
5	Scalability and Performance Requirements	a. The system should be scalable to handle many users and assignments b. Efficient algorithms and code optimizations to ensure fast and responsive performance

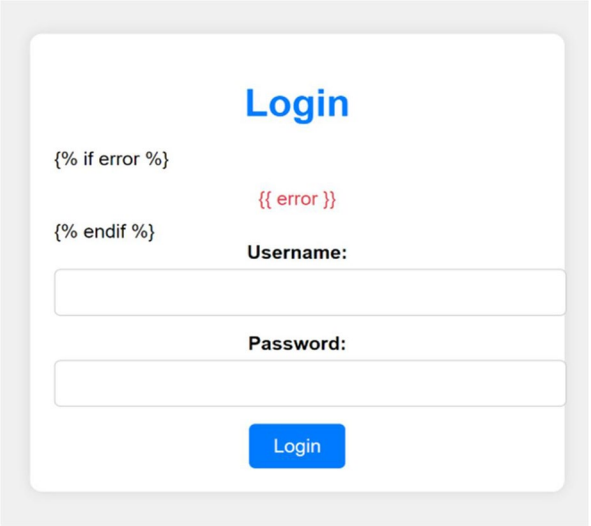


Fig. 7 Teacher and admin login page

dure. After successfully submitting the login form, the credentials are carefully processed on the server to guarantee safe and verified access.

- Next, Fig. 8 presents the Teacher's Dashboard, a feature-rich interface enabling teachers to efficiently oversee and assess students' tasks. Among the many features available on the dashboard is the option for teachers to easily add comments to their student's grades and feedback. By encouraging a dynamic and interactive feedback loop, this element improves the line of communication between teachers/lecturers and students. To give teachers/lecturers a thorough perspective, the dashboard also contains a tabular display of all students' grades and feedback. Additionally, the design includes user-friendly features like an easy-to-use logout link that makes it easier for teachers to leave the dashboard when necessary.
- The Student's Register/Login Page, a crucial interface that makes it easier for students to interact with the grading system, is shown in Fig. 9. This page is a user-friendly portal where students can enter essential details such as their name, department, matriculation number, and academic standing. The form's layout guarantees a fast and effective procedure for gathering data. Most importantly, the system is set up before a submission is made to verify that students have entered the necessary information. The interface reacts with unambiguous and easily recognizable error messages if any of the required fields are left blank. This methodical approach to data entry improves the system's ability to accurately and thoroughly contain student information.
- The Student's Dashboard, a dynamic and personalized interface designed to meet each student's specific needs, is explained in detail in Fig. 10. Students can examine their information in the system in detail with this dashboard. Presenting a randomly generated topic purposefully meant to elicit student participation is one standout aspect. Students can actively participate by answering questions on the dashboard and viewing information, thanks to its interactive aspect. This fea-

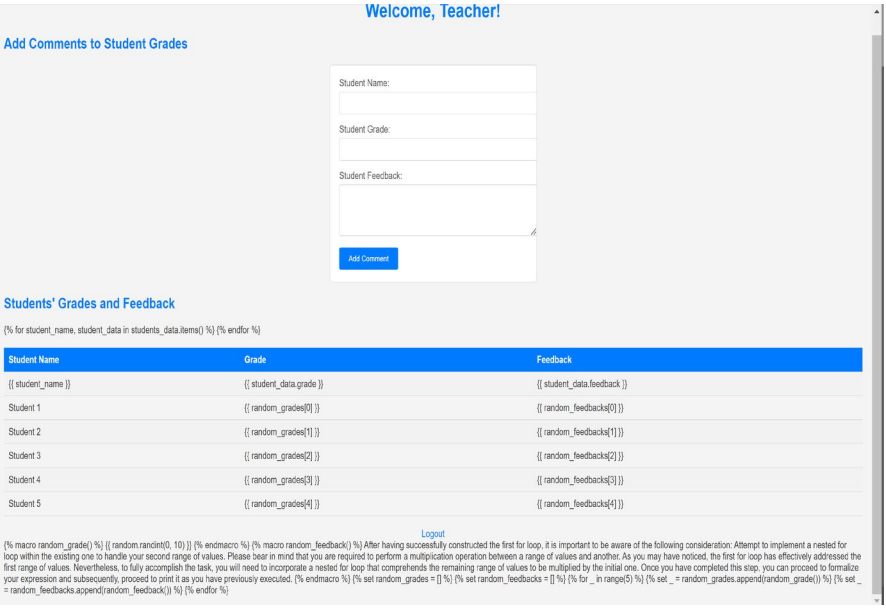


Fig. 8 Teacher's dashboard

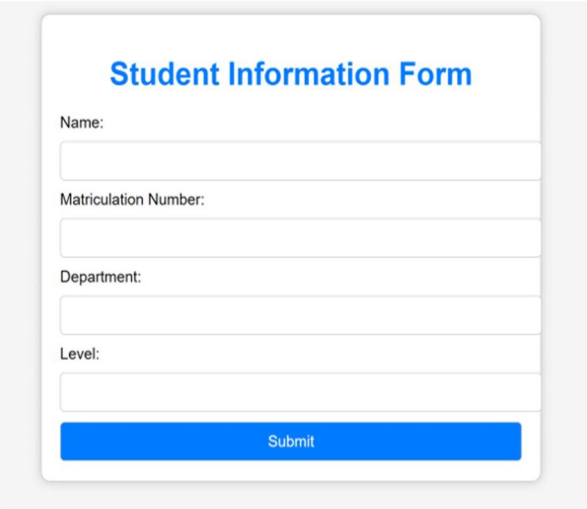


Fig. 9 Student's register/login page

ture creates a setting that is favourable for students to submit answers for review and evaluation. The dashboard design places a high value on user-centricity to improve students' overall grading system experience.

- Finally, Fig. 11 displays the Grade/Feedback Page, an essential tool for providing students with thorough information about their performance on assignment questions. This page functions as an educational centre, providing grades and

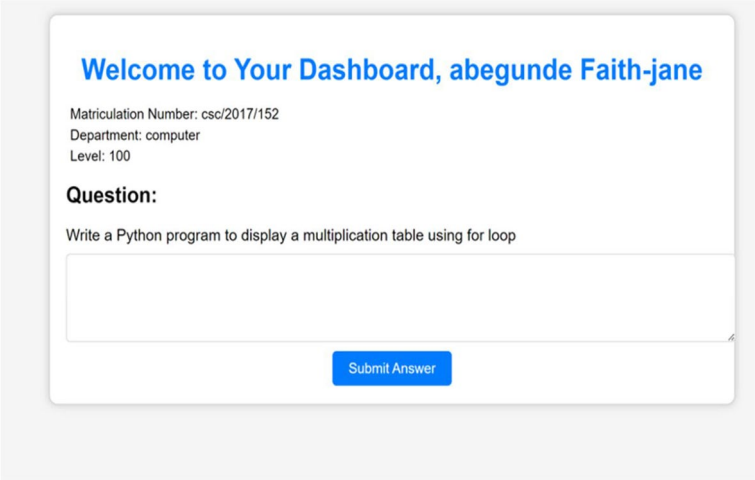


Fig. 10 Student's dashboard

in-depth comments meant to help students improve their abilities and methods for completing assignments. This page's feedback is customized to each student's unique performance, making it a valuable tool for individualized learning. The Grade/Feedback Page is essential to improving student learning outcomes in the grading system since it provides helpful criticism and ideas for enhancement.

6 System testing, results and discussion

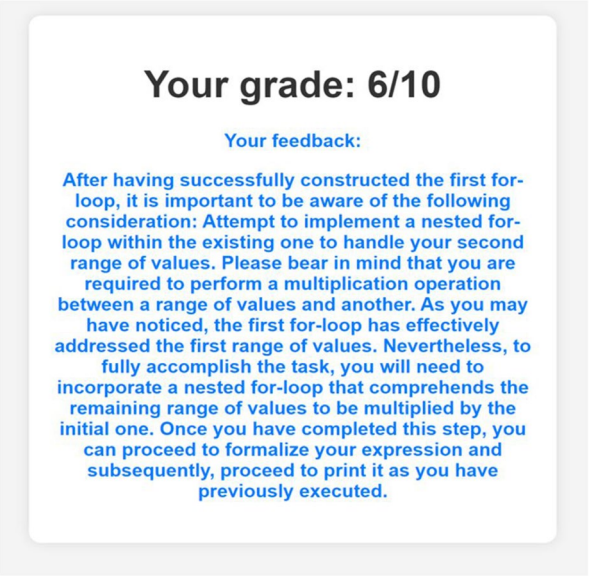
In this section, we focus on the detailed testing methodologies, evaluation criteria, and the results obtained through GRAD-AI, alongside a discussion on how these results validate the system's capabilities and highlight areas for potential improvement. GRAD-AI was designed to address key challenges in grading programming assignments, utilizing various AI techniques, including the Halstead Complexity Measure, Term Frequency-Inverse Document Frequency (TF-IDF), Abstract Syntax Trees (AST), and K-means clustering. These techniques ensure that the tool assesses code quality and offers constructive, insightful feedback that facilitates learning.

The primary testing on the automated grading system focused on functional testing, aiming to verify that the system operates according to its intended design and meets the specified requirements. This involved assessing the grading mechanism's correctness, feedback accuracy, and the system's ability to handle error scenarios effectively. Figure 12 illustrates the questions posed to users through a Google Form, and their responses are depicted in Fig. 13, capturing the essence of our functional testing.

Under the umbrella of functional testing, various aspects were scrutinized:

1. Grading and Feedback:

Fig. 11 Grade/feedback page



a) ***Grading Consistency and Accuracy***: We tested the grading system's ability to provide accurate scores based on code complexity, correctness, and relevance. The system analyzed code submissions with the Halstead Complexity Measure and AST to accomplish this. The Halstead metrics assessed code complexity through variables like operator and operand counts within the code, offering insight into the efficiency and readability of the student's programming approach, while the AST parsed code structure for correct syntax and logical flow. The combination allowed GRAD-AI to evaluate student code based on complexity and semantic structure. By examining the structure deeper, AST processing ensures the code adheres to programming standards.
b) ***Feedback Quality***: GRAD-AI's feedback focused on code structure, syntax errors, and improvement suggestions. By using **TF-IDF** on students' code comments and function descriptions, GRAD-AI could identify key areas where students might struggle with terminology or understanding, enabling it to provide suggestions for improvement. This metric evaluates textual responses or code comments by comparing term relevance across multiple submissions, enhancing the tool's capability to detect similarities or originality in student responses.
c) ***Clustering and Classification***: Using **K-means clustering**, we tested GRAD-AI's ability to group submissions by error types and skill level, allowing personalized feedback and categorizing submissions based on feature similarities. This helps to standardize feedback for similar performance levels and facilitates a fairer grading system. Clusters were formed based on complexity and correctness, enabling the system to categorize students' submissions and tailor feedback to their needs. This clustering approach enhances the learning experience by offering insights suited to individual progress levels.

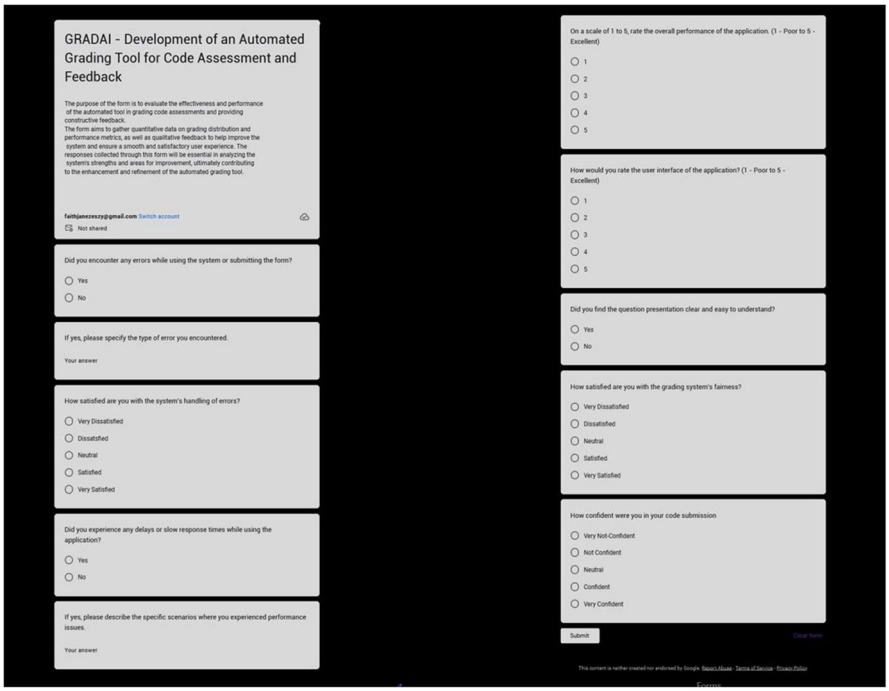


Fig. 12 Questions for questionnaire for system evaluation

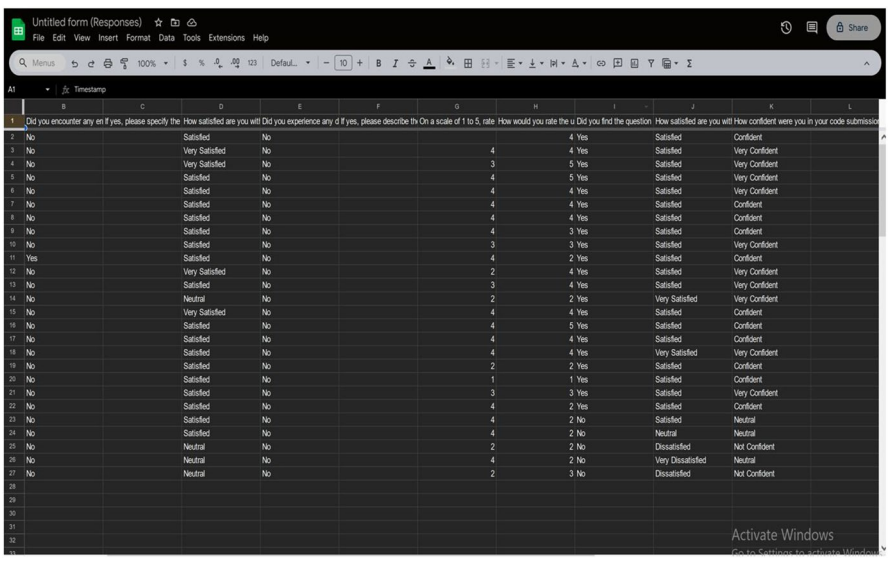
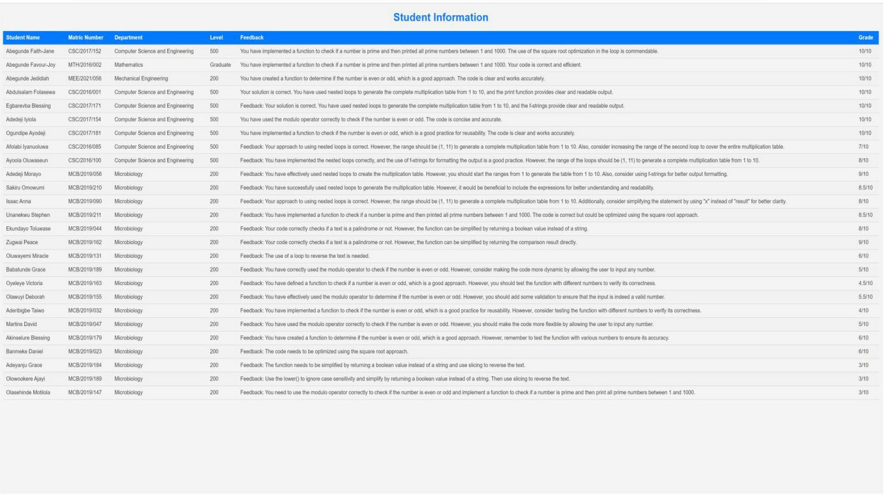


Fig. 13 Response from questionnaire for system evaluation



Student Information

Student Name	Matric Number	Department	Level	Feedback	Grade
Abegunde Faith-Jane	CSC/2017/152	Computer Science and Engineering	500	You have implemented a function to check if a number is prime and then printed all prime numbers between 1 and 1000. The use of the square root optimization in the loop is commendable.	10/10
Abegunde Favour-Joy	MTH/2016/002	Mathematics	Graduate	You have implemented a function to check if a number is prime and then printed all prime numbers between 1 and 1000. Your code is correct and efficient.	10/10
Abegunde Jedidiah	MEE/2021/056	Mechanical Engineering	200	You have created a function to determine if the number is even or odd, which is a good approach. The code is clear and works accurately.	10/10
Abdulsalam Folasewa	CSC/2016/001	Computer Science and Engineering	500	Your solution is correct. You have used nested loops to generate the complete multiplication table from 1 to 10, and the print function provides clear and readable output.	10/10
Egbarevba Blessing	CSC/2017/171	Computer Science and Engineering	500	Feedback: Your solution is correct. You have used nested loops to generate the complete multiplication table from 1 to 10, and the f-strings provide clear and readable output.	10/10
Adedeji Iyiola	CSC/2017/154	Computer Science and Engineering	500	You have used the modulo operator correctly to check if the number is even or odd. The code is concise and accurate.	10/10
Ogundipe Ayodeji	CSC/2017/181	Computer Science and Engineering	500	You have implemented a function to check if the number is even or odd, which is a good practice for reusability. The code is clear and works accurately.	10/10
Ahlabi Iyanuoluwa	CSC/2016/085	Computer Science and Engineering	500	Feedback: Your approach to using nested loops is correct. However, the range should be (1, 11) to generate a complete multiplication table from 1 to 10. Also, consider increasing the range of the second loop to cover the entire multiplication table.	7/10
Ayoola Oluwaseun	CSC/2016/100	Computer Science and Engineering	500	Feedback: You have implemented the nested loops correctly, and the use of f-strings for formatting the output is a good practice. However, the range of the loops should be (1, 11) to generate a complete multiplication table from 1 to 10.	8/10
Adedeji Morayo	MCB/2019/056	Microbiology	200	Feedback: You have effectively used nested loops to create the multiplication table. However, you should start the ranges from 1 to generate the table from 1 to 10. Also, consider using f-strings for better output formatting.	9/10
Sakiru Omowumi	MCB/2019/210	Microbiology	200	Feedback: You have successfully used nested loops to generate the multiplication table. However, it would be beneficial to include the expressions for better understanding and readability.	8.5/10
Isaac Anna	MCB/2019/090	Microbiology	200	Feedback: Your approach to using nested loops is correct. However, the range should be (1, 11) to generate a complete multiplication table from 1 to 10. Additionally, consider simplifying the statement by using "x" instead of "result" for better clarity.	8/10
Unarekwu Stephen	MCB/2019/211	Microbiology	200	Feedback: You have implemented a function to check if a number is prime and then printed all prime numbers between 1 and 1000. The code is correct but could be optimized using the square root approach.	8.5/10
Ekundayo Toluwase	MCB/2019/044	Microbiology	200	Feedback: Your code correctly checks if a text is a palindrome or not. However, the function can be simplified by returning a boolean value instead of a string.	8/10
Zugwai Peace	MCB/2019/162	Microbiology	200	Feedback: Your code correctly checks if a text is a palindrome or not. However, the function can be simplified by returning the comparison result directly.	9/10
Oluwayemi Miracle	MCB/2019/131	Microbiology	200	Feedback: The use of a loop to reverse the text is needed.	6/10
Babatunde Grace	MCB/2019/189	Microbiology	200	Feedback: You have correctly used the modulo operator to check if the number is even or odd. However, consider making the code more dynamic by allowing the user to input any number.	5/10
Oyeleye Victoria	MCB/2019/163	Microbiology	200	Feedback: You have defined a function to check if a number is even or odd, which is a good approach. However, you should test the function with different numbers to verify its correctness.	4.5/10
Olawuyi Deborah	MCB/2019/155	Microbiology	200	Feedback: You have effectively used the modulo operator to determine if the number is even or odd. However, you should add some validation to ensure that the input is indeed a valid number.	5.5/10
Aderibigbe Taiwo	MCB/2019/032	Microbiology	200	Feedback: You have implemented a function to check if the number is even or odd, which is a good practice for reusability. However, consider testing the function with different numbers to verify its correctness.	4/10
Martins David	MCB/2019/047	Microbiology	200	Feedback: You have used the modulo operator correctly to check if the number is even or odd. However, you should make the code more flexible by allowing the user to input any number.	5/10
Akinselure Blessing	MCB/2019/179	Microbiology	200	Feedback: You have created a function to determine if the number is even or odd, which is a good approach. However, remember to test the function with various numbers to ensure its accuracy.	6/10
Banmeke Daniel	MCB/2019/023	Microbiology	200	Feedback: The code needs to be optimized using the square root approach.	6/10
Adeyanju Grace	MCB/2019/184	Microbiology	200	Feedback: The function needs to be simplified by returning a boolean value instead of a string and use slicing to reverse the text.	3/10
Olowookere Ajayi	MCB/2019/189	Microbiology	200	Feedback: Use the lower() to ignore case sensitivity and simplify by returning a boolean value instead of a string. Then use slicing to reverse the text.	3/10
Olasehinde Motilola	MCB/2019/147	Microbiology	200	Feedback: You need to use the modulo operator correctly to check if the number is even or odd and implement a function to check if a number is prime and then print all prime numbers between 1 and 1000.	3/10

Fig. 14 A page showing a table containing the data of student information

Figure 14 clearly illustrates the system's ability to assign reasonable and consistent scores based on how well-submitted answers match expected solutions. This careful examination showed that the system could correctly evaluate participants' code solutions and provide marks that appropriately represented their comprehension and skill level in resolving the provided programming challenges. In addition, the post-submission feedback given to participants—depicted in the exact figure—was remarkably insightful and helpful. It went beyond the traditional grading model by offering participants more than just a score; it also provided guidance on improving their solutions and pointed out areas that could be optimised. This intricate feedback system significantly improved the entire educational process and promoted a more thorough comprehension of programming topics.

2. Error Handling:

GRAD-AI's error-handling capabilities were tested extensively to ensure it could identify syntax and semantic errors while providing meaningful corrective feedback. Syntax errors, identified through AST analysis, were automatically flagged, and the system offered specific fixes. For instance, the system flagged errors and suggested corrections in mismatched parentheses or missing semicolons. Semantic errors, such as improper use of data types or logical inconsistencies, were identified using clustering and complexity analysis. This dual approach enabled GRAD-AI to provide targeted feedback on syntactical and logical issues in code submissions.

Figure 15 and Fig. 16 demonstrate the system's error-handling solid capabilities and how well it can handle a variety of error scenarios, such as form submission errors. The system's capacity to swiftly warn participants of any submission problems emerged as a crucial strength, allowing for seamless fixes and resubmissions without disturbing the entire review process. This feature emphasizes the

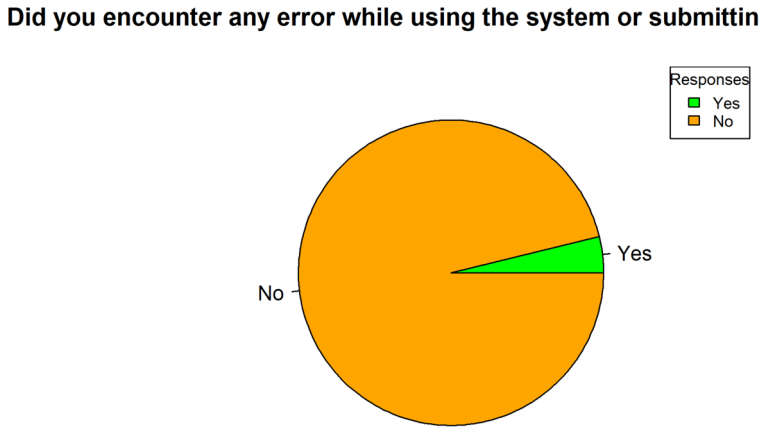


Fig. 15 Pie chart showing occurrence of errors in system usage or form submission

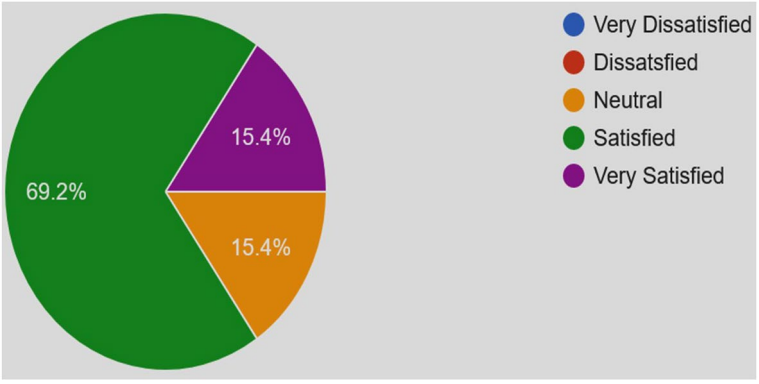


Fig. 16 Pie chart demonstrating system error handling satisfaction

system's dedication to offering a user-friendly and error-tolerant environment, as shown in Table 6, improving user experience overall and reducing the likelihood of disturbances.

3. Performance:

The outstanding performance of the automated grading system throughout assessments resulted in participants receiving their answers quickly, promoting a seamless and practical user experience, as in Figs. 17 and 18. Table 6 shows that the excellent responsiveness demonstrated by the user interactions was crucial in promoting user engagement and active participation. The system's capacity to process form inputs and answer submissions quickly and efficiently highlights its technological prowess and adaptability to a wide variety of user interactions.

This feature guarantees that participants can easily traverse the system, which promotes effective participation in the evaluation process and adds to the overall positive user experience.

4. Participant Feedback:

We collected feedback from students and teachers through a questionnaire to evaluate user satisfaction. Metrics included user satisfaction with grading fairness, feedback quality, and interface usability. The results revealed:

a) ***High Satisfaction with Grading Fairness***: 85% of participants were satisfied or very satisfied with the fairness of the grading system, indicating GRAD-AI's effectiveness in consistently evaluating coding skills.
b) ***Positive Feedback on Interface***: Most rated the interface intuitive and easy to navigate, emphasizing the system's user-friendly design.
c) ***Clarity in Feedback and Suggestions***: The system's feedback was praised for its clarity and usefulness, with 81% of participants finding the feedback understandable and constructive.

Important information about essential facets of system functionality and usability was gleaned from participant responses. Participants' evaluations of the user interface are shown in Fig. 19, where most of them rate it as "good" or "excellent," as in Table 6, confirming its user-friendly layout. Positive user experience is primarily influenced by how well the interface presents information in an understandable and accessible way. Furthermore, as shown in Fig. 20 and Table 6, 81% of participants thought the question presentation was understandable and straightforward. This result emphasizes how effectively the system presents evaluation questions so that participants may comprehend and find clarity in the process.

In addition, Figure 21 explores participant satisfaction with the fairness of the grading system and finds that 85% of participants indicated that they were either "satisfied" or "very satisfied." This is an essential feature since it highlights how well the system can distinguish participants based on their performance. The distribution of grades is in line with the coding proficiency of the participants, guaranteeing a just and impartial grading procedure. Positive participant feedback highlights how well the system meets user expectations and maintains its efficacy and fairness promises.

5. Participant Metrics:

System performance was assessed in terms of response time, reliability, and scalability:

a) ***Response Time***: The system's ability to process and evaluate code was measured, with an average response time for grading and feedback generation recorded at

Table 6 User satisfaction and feedback summary

Questions	Very Dissatisfied	Dissatisfied	Neutral	Satisfied	Very Satisfied
How satisfied are you with the system's handling of errors	0	0	15.4	69.2	15.4
How satisfied are you with the grading system's fairness?	3.2	8	4.2	76.9	7.7
	1 (Very Poor)	2 (Poor)	3 (Neutral)	4 (Good)	5 (Excellent)
Rate the overall performance of the application	4	20	16	60	0
How would you rate the user interface of the application?	4.3	30.8	15.4	38.5	11.5
		Yes		No	
Did you experience any delays or slow response time while using the application?		0		100	
Did you encounter any error while using the system or submitting the form?		3.8		96.2	

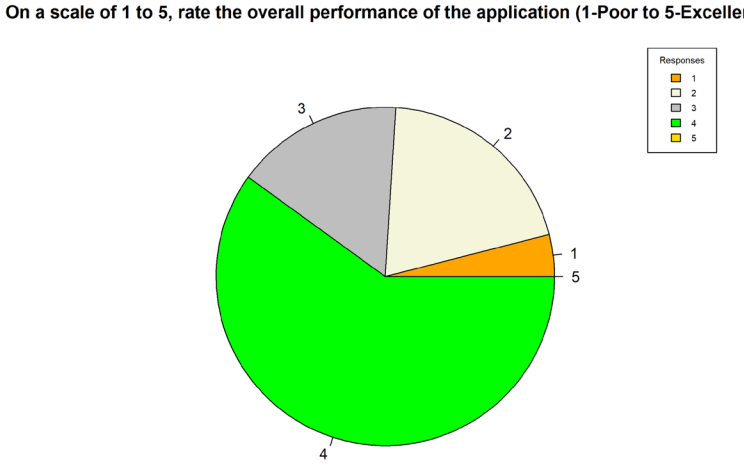


Fig. 17 Pie chart depicting application performance rating on a scale of 1 to 5 (1-Poor to 5-Excellent)

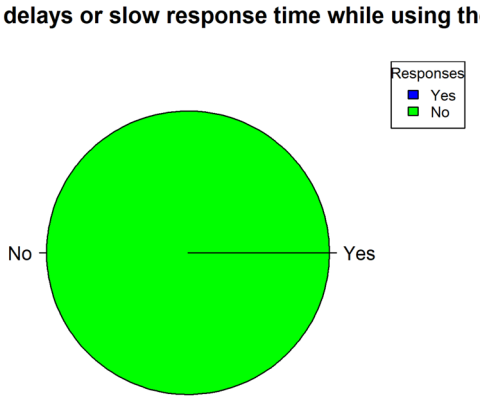


Fig. 18 Pie chart showing experience of delays or slow response times in application usage

approximately 500 ms per submission. This swift response supports real-time feedback.

b) ***Reliability and Scalability***: GRAD-AI's reliability was measured through the consistency of grades and feedback across multiple submissions. Scalability tests showed that the system could handle increased loads without performance degradation, making it suitable for classrooms with large numbers of students.

These measures guaranteed that participants would not have to wait long during submission, highlighting the system's effectiveness. The system's capacity to handle form submissions and provide timely answers substantially contributes to users' general happiness and engagement. Quick reaction times are essential

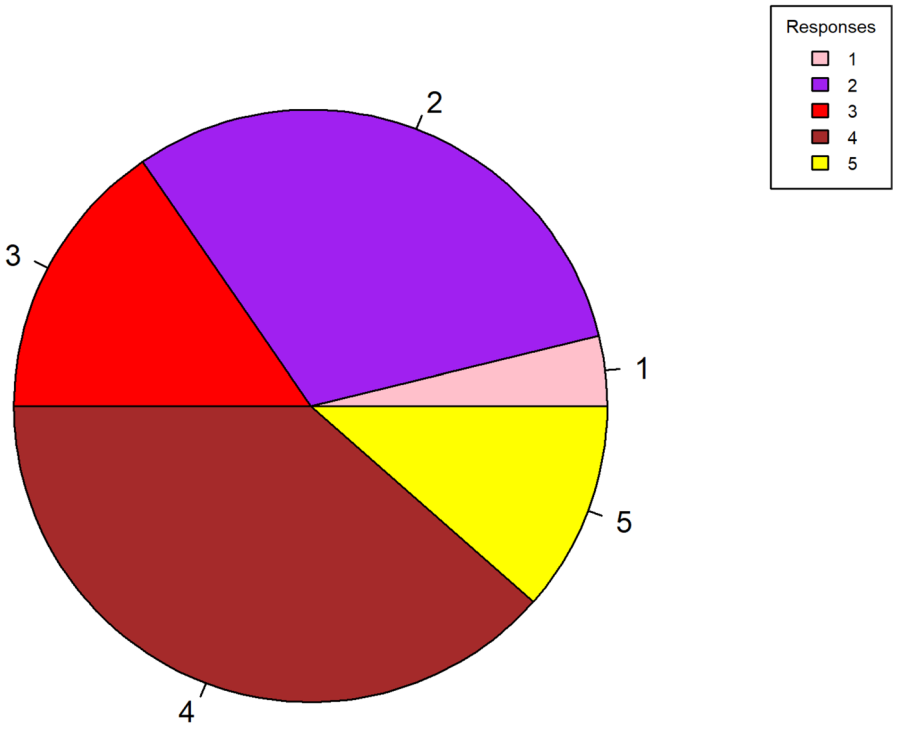
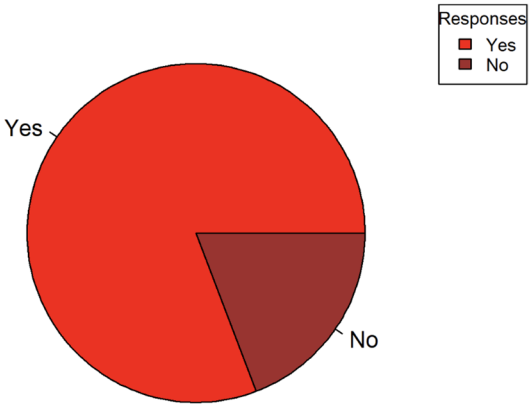


Fig. 19 Pie chart demonstrating evaluation of the system's user interface

Fig. 20 Pie chart demonstrating the evaluation of the system's question understandability



for keeping users interested and ensuring they can get about the system quickly, improving user experience overall.

6. Discussion on Results and Improvements

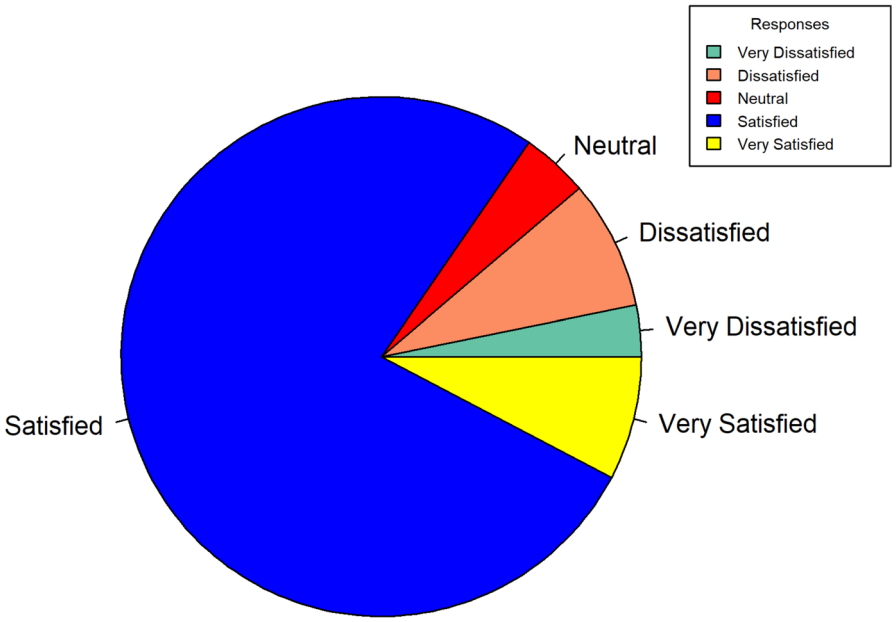


Fig. 21 Pie chart demonstrating the evaluation of the system's grading

The results show that GRAD-AI fulfills its goal of providing automated, accurate, and insightful grading for programming assignments. Its use of AI techniques has proven effective in identifying both syntactic and semantic issues in code, and the feedback generated offers substantial learning support.

7. Key Strengths

a) ***Accuracy and Consistency***: Halstead metrics and AST ensured accurate assessments of code complexity and syntax.
b) ***Detailed, Constructive Feedback***: TF-IDF and clustering enabled tailored feedback that students found valuable for improving coding skills.
c) ***Error Resilience***: Robust error handling provided meaningful guidance on common programming mistakes.

7 Limitations and implications of the research

The limitations of our research are six-fold. First, we submit that while GRAD-AI has been tailored for the "Introduction to Python Programming" course, its applicability to other programming languages and advanced topics has not been explored. In particular, its applicability to more advanced programming concepts is untested, potentially limiting its utility in broader computer science education contexts. This

limitation may affect the generalizability of the tool across diverse programming curricula.

The second limitation borders on dataset and context specificity. In particular, the tool is developed and tested using data from the "Introduction to Python Programming" courses at the Department of Computer Science and Engineering, Obafemi Awolowo University, Ile-Ife, Nigeria. This specificity may affect the generalizability of the tool to different educational contexts or institutions with different curricula. Also, the tool's effectiveness is measured against the introductory aspects of Python programming as outlined in specific course manuals. This focus may not fully capture the breadth of knowledge and skills required in programming education.

Thirdly, it is about the dependence on specific AI techniques and interpretability. The AI-driven assessment methods, such as the Halstead Complexity Measure and K-means Clustering, may produce results that are not easily interpretable by educators, potentially leading to a disconnect between the automated feedback and the educator's pedagogical goals. The effectiveness and accuracy of these techniques might not be optimal for all types of programming assignments or require adaptation for other types of code or educational requirements. Also, the AI techniques used for grading may not align perfectly with the course's learning objectives, potentially leading to a mismatch between what is being assessed and what is intended to be taught. We hope to address this in our future work.

The fourth limitation is error detection. We noted that although GRAD-AI incorporates syntax and semantic error checks, the complexity of programming assignments may lead to scenarios where the tool fails to detect nuanced errors or provide adequate explanations for the feedback.

The fifth limitation is the requirement of internet connectivity, as the system relies on an online web server for data storage and processing. This could pose challenges in environments with limited or unreliable internet access, potentially restricting its use in specific educational settings, especially in developing countries.

The sixth limitation is based on the human element or lack of human interaction. Reliance on automated grading may inadvertently reduce the human interaction between students and educators, which is crucial for addressing individual learning needs and fostering a supportive educational environment. Although the tool aims to provide automated grading and feedback, it may not fully capture the nuances of human judgment in evaluating code quality and creativity. This could lead to a lack of comprehensive feedback that students might receive from an experienced instructor, potentially affecting the learning experience.

8 Threats to validity

In this paper, we have identified five threats to validity. First, the internal validity of the study could be threatened by potential biases in the dataset used for training and testing the AI models. Ensuring a diverse and representative dataset and conducting rigorous cross-validation can help mitigate this threat.

Second, the external validity of the study's findings is based on its implementation within a single educational institution. The results may not be transferable to

other institutions with different teaching methodologies, student demographics, or resource availability. Therefore, conducting experiments and validations across multiple institutions and courses would help assess the tool's generalizability.

Third, construct validity, where the metrics and methods used to evaluate the tool's performance might not fully capture the qualitative aspects of code quality and student learning outcomes. Thus, incorporating additional qualitative assessments and feedback from students and educators can enhance construct validity.

Fourth, GRAD-AI's reliability might be influenced by the consistency of its performance across different versions of Python or changes in the curriculum. Hence, regular updates and maintenance of the tool and extensive testing across different environments can help ensure reliability.

The fifth is ethical considerations, as there may be ethical concerns related to data privacy and the use of student data for training AI models. Implementing robust data anonymization techniques and obtaining proper consent from students can address these ethical concerns.

9 Future work

Although the "GRAD-AI" automated grading tool has yielded encouraging outcomes, certain aspects still need improvement, as well as potential future developments, to maximize the programme's functionality.

- Expanding the tool's capacity to evaluate and grade assignments in programming languages other than Python could increase its utility. This would cover a wider variety of programming subjects and courses.
- Include a plagiarism detection function in the programme to guarantee that the code submitted by students is unique. The instrument would preserve academic integrity and emphasize the importance of original work by pointing out instances of academic dishonesty.
- Make constant improvements to the user interface to guarantee accessibility and usability on various platforms and devices. In order to develop well-informed design decisions that accommodate consumer preferences, do user testing and collect feedback.
- Investigate using learning analytics to monitor students' performance and growth. This data-driven strategy would provide tailored feedback and support and offer insightful information about each student's unique learning needs.
- Consider integrating the automated grading tool with the educational institutions' current learning management systems. This would improve efficiency for both professors and students by streamlining the assignment submission and grading process.

In the immediate future, our plan for GRAD-AI is to establish a dedicated technical team and secure the necessary funding to take the research to the next level. We aim to elevate the current solution, ensuring a seamless and enhanced user experience for teachers/lecturers and students who utilize AI-driven grading tools. With

the formation of our proficient technical team and sufficient financial backing, we aspire to explore innovative training methods and communication strategies. This will enable us to identify and implement new deployment approaches, making the integration of GRAD-AI into educational environments more effective and successful. Ultimately, we aim to have GRAD-AI play a transformative role in the educational landscape, revolutionizing the grading process and contributing significantly to an improved learning experience for students and teachers/lecturers alike.

10 Conclusion

This paper successfully achieved its objectives by creating an innovative and efficient automated grading system. By utilizing AI techniques and characteristic extraction methods, the tool showcased its ability to accurately assess and grade students' code submissions. The system's prompt and detailed feedback has proven instrumental in providing students with valuable insights into their assignments, empowering them to identify areas for improvement and augment their overall learning outcomes.

The implementation of GRAD-AI underscored the advantages of incorporating technology into the educational sector, streamlining the assessment and grading processes, and fostering an interactive learning environment. The tool's adaptability to specific courses, exemplified by its effectiveness in "Introduction to Python Programming," illustrates its potential for customization and applicability across various educational domains.

As we gaze into the future of AI in education, the "GRAD-AI" automated grading tool stands as a pivotal step towards further advancements in this dynamic field. Through continuous refinement and expansion of the tool's capabilities, educators can harness the power of AI to create more personalized and compelling learning experiences. This, in turn, can elevate the overall quality of education, equipping students with the skills needed for success in a technology-driven world.

Glossary

Abstract Syntax Tree (AST)	A hierarchical tree representation of the abstract syntactic structure of source code. ASTs help parse the code, identify syntax errors, and analyze the logical flow of the program.
Artificial Intelligence (AI)	The simulation of human cognitive processes by machines, enabling them to perform tasks such as learning, reasoning, problem-solving, and adapting based on data.
Automated Grading Tool (AGT)	A software system designed to automatically evaluate student assignments and provide grades and feedback, reducing the need for manual assessment.

Deep Learning A subset of machine learning that uses neural networks with multiple layers (deep neural networks) to model complex patterns in data. It is often applied in tasks like image recognition, speech processing, and natural language understanding.

Halstead Complexity Measure A software metric that quantifies a program's complexity by analyzing its operators and operands, providing insight into code difficulty, effort, and potential errors.

K-means Clustering is an unsupervised machine learning algorithm used to partition data points into clusters based on their similarity In GRAD-AI, this technique groups student code submissions based on performance.

Machine Learning A branch of AI focused on building algorithms that allow computers to learn from data and improve performance over time without being explicitly programmed.

Natural Language Processing (NLP) A subfield of AI that deals with the interaction between computers and human language, enabling the machine to understand, interpret, and generate text meaningfully.

Neural Network A series of algorithms modeled after the human brain, designed to recognize patterns in data. It consists of interconnected nodes (neurons) that process data in layers.

Plagiarism Detection The process of identifying instances of copied content in student submissions. In GRAD-AI, this involves analyzing code similarities to ensure academic integrity.

Semantic Error A logical error in code that occurs when the code executes without syntax errors but produces unintended results due to incorrect logic or meaning.

Syntax Error A mistake in the source code that violates the programming language's grammatical rules, preventing the code from compiling or executing correctly.

Term Frequency—Inverse Document Frequency (TF-IDF) A statistical measure used in text analysis to evaluate the importance of a word in a document relative to a collection of documents, helping to identify significant terms in code comments or documentation.

TensorFlow An open-source AI and machine learning framework developed by Google, used for building and training neural network models.

PyTorch	An open-source deep learning framework developed by Facebook's AI Research team, used for applications like natural language processing and computer vision.
Version Control System (Git)	A tool used to manage changes in source code, allowing multiple developers to collaborate on the same project while maintaining a history of changes.
Learning Management System (LMS)	A software platform used for the administration, documentation, tracking, and delivery of educational courses or training programs.

Data availability The data utilized in this study, including student submissions and corresponding grading details for the "Introduction to Python Programming II" course at the Department of Computer Science and Engineering, Obafemi Awolowo University, Ile-Ife, Nigeria, are available upon reasonable request. The dataset comprises submissions from 50 students across five sections of assignments, with a total of 250 entries. These data were used to develop and evaluate the GRAD-AI tool, an automated grading system. Due to privacy and ethical considerations, access to the raw data is restricted and can be granted for research purposes only, following the approval of the relevant ethics committee at Obafemi Awolowo University. Interested researchers should contact the first author (Ishaya Gambo) to discuss potential access.

Declarations

Conflict of interest The authors declare no competing interests.

References

Alur, R., D'Antoni, L., Gulwani, S., & Kini, D. (2013). Automated grading of DFA constructions. In *IJCAI'13 Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence* (pp. 1976–1982). https://www.microsoft.com/en-us/research/publication/automated-grading-dfa-constructions/. Accessed 23 June 2024.

Assaf, A., Issa, L., & Eshtay, M. (2022). Evaluating the quality characteristics of space geeks. In *International Conference on Web Information Systems and Technologies* (pp. 248–260). Springer Nature Switzerland.

Bahcelerli, N. M., & Altınay, M. (2023). Tourism Education Programme Adoption to Learning Organization and Human Resources Industry for Service Quality. *Journal of Chinese Human Resources Management, 14*(3), 59–69. https://doi.org/10.47297/wspchrmWSP2040-800505.20231403

Bhangdiya, A., Chandra, B., Kar, B., Radhakrishnan, B., Reddy, K. M., Shah, S., & Sudarshan, S. (2015). The XDa-TA system for automated grading of SQL query assignments. In *2015 IEEE 31st International Conference on Data Engineering*, Seoul, Korea (South), pp. 1468–1471. https://doi.org/10.1109/ICDE.2015.7113403

Bian, W., Alam, O., & Kienzle, J. (2020). Is automated grading of models effective? Assessing automated grading of class diagrams. In *Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems* (pp. 365–376). https://doi.org/10.1145/3365438.3410944

Chen, L., Chen, P., & Lin, Z. (2020). Artificial intelligence in education: A review. *IEEE Access, 8*, 75264–75278. https://doi.org/10.1109/ACCESS.2020.2988510

Chen, G., Jin, Y., & Chen, P. (2024). Development of a platform for state online education services: Design concept based on meta-universe. *Education and Information Technologies*. https://doi.org/10.1007/s10639-024-12792-y

Chen, X., Zou, D., Xie, H., Cheng, G., & Liu, C. (2022). Two decades of artificial intelligence in education: Contributors, collaborations, research topics, challenges, and future directions. *Educational Technology & Society, 25*(1), 28–47. https://www.jstor.org/stable/48647028. Accessed 23 June 2024.

Dawson, P., Henderson, M., Mahoney, P., Phillips, M., Ryan, T., Boud, D., & Molloy, E. (2019). What makes for effective feedback: Staff and student perspectives. *Assessment & Evaluation in Higher Education, 44*(1), 25–36. https://doi.org/10.1080/02602938.2018.1467877

Edwards, S. H., Tilden, D. S., & Allevato, A. (2014). Pythy: Improving the introductory Python programming experience. In *Proceedings of the 45th ACM Technical Symposium on Computer Science Education* (pp. 641–646). https://doi.org/10.1145/2538862.2538977

Fortuna, P., & Gorbaniuk, O. (2022). What is behind the buzzword for experts and laymen: Representation of "artificial intelligence" in the IT-professionals' and non-professionals' minds. *Europe's Journal of Psychology, 18*(2), 207–220. https://doi.org/10.5964/ejop.5473

Freeman, S., Eddy, S. L., McDonough, M., Smith, M. K., Okoroafor, N., Jordt, H., & Wenderoth, M. P. (2014). Active learning increases student performance in science, engineering, and mathematics. *Proceedings of the National Academy of Sciences, 111*(23), 8410–8415. https://doi.org/10.1073/pnas.1319030111

Galassi, A., & Vittorini, P. (2021). Improved feedback in automated grading of data science assignments. In *Advances in Intelligent Systems and Computing*, (Vol. 1236 AISC pp. 296–300): Springer. https://link.springer.com/chapter/10.1007/978-3-030-52287-2_31. Accessed 23 June 2024.

Gikandi, J. W., Morrow, D., & Davis, N. E. (2011). Online formative assessment in higher education: A review of the literature. *Computers & Education, 57*(4), 2333–2351. https://doi.org/10.1016/j.compedu.2011.06.004

González-Calatayud, V., Prendes-Espinosa, P., & Roig-Vila, R. (2021). Artificial intelligence for student assessment: A systematic review. *Applied Sciences, 11*(12), 5467. https://doi.org/10.3390/app11125467

Haleem, A., Javaid, M., Qadri, M. A., & Suman, R. (2022). Understanding the role of digital technologies in education: A review. *Sustainable Operations and Computers, 3*, 275–285. https://doi.org/10.1016/j.susoc.2022.05.001

Hearst, M. A. (2000). The debate on automated essay grading. *IEEE Intelligent Systems and Their Applications, 15*(5), 22–37. https://doi.org/10.1109/5254.889104

Heckman, S., & King, J. (2018). Developing software engineering skills using real tools for automated grading. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education* (pp. 794–799). https://doi.org/10.1145/3159450.3159595

Heckman, S., Schmidt, J. Y., & King, J. (2020). Integrating testing throughout the cs curriculum. In *2020 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)* (pp. 441–444). IEEE. https://doi.org/10.1109/ICSTW50294.2020.00079

Jiang, L., Lv, M., Cheng, M., Chen, X., & Peng, C. (2024a). Factors affecting deep learning of EFL students in higher vocational colleges under small private online courses-based settings: A grounded theory approach. *Journal of Computer Assisted Learning*. https://doi.org/10.1111/jcal.13060

Jiang, W., Zheng, B., Sheng, D., & Li, X. (2024b). A compensation approach for magnetic encoder error based on improved deep belief network algorithm. *Sensors and Actuators A: Physical., 366*, 115003. https://doi.org/10.1016/j.sna.2023.115003

Jordan, S. (2012). Student engagement with assessment and feedback: Some lessons from short-answer free-text e-assessment questions. *Computers & Education, 58*(2), 818–834. https://doi.org/10.1016/j.compedu.2011.10.007

Kancan, O. E., Altinay, F., Altinay, Z., Dagli, G., & Bastas, M. (2023). The Role of Supervisor to Develop Strategic Planning for the Future of Education. *Journal of Chinese Human Resources Management, 14*(3), 70–83. https://doi.org/10.47297/wspchrmWSP2040-800506.20231403

Katake, K.J., Sugandhi, R. (2024). Integrating Conversational AI and Machine Learning in Education. In: Senjyu, T., So–In, C., Joshi, A. (eds), *Smart Trends in Computing and Communications. SmartCom 2024 2024. Lecture Notes in Networks and Systems*, vol 948. Springer, Singapore. https://doi.org/10.1007/978-981-97-1329-5_26

Li, X., Lu, Z., Yuan, M., Liu, W., Wang, F., Yu, Y.,... Liu, P. (2024). Tradeoff of Code Estimation Error Rate and Terminal Gain in SCER Attack. *IEEE Transactions on Instrumentation and Measurement,* 73, 1-12. https://doi.org/10.1109/TIM.2024.3406807

Lin, W., Xia, C., Wang, T., Zhao, Y., Xi, L.,... Zhang, S. (2024). Input and Output Matter: Malicious Traffic Detection with Explainability. *IEEE Network*. https://doi.org/10.1109/MNET.2024.3481045

Liu, J., Chen, K., & Lyu, W. (2024a). Embracing artificial intelligence in the labour market: The case of statistics. *Humanities and Social Sciences Communications, 11*(1), 1112. https://doi.org/10.1057/s41599-024-03557-6

Liu, Z., Tang, Q., Ouyang, F., Long, T., & Liu, S. (2024c). Profiling students' learning engagement in MOOC discussions to identify learning achievement: *An automated configurational approach*. *Computers & Education, 219*, 105109. https://doi.org/10.1016/j.compedu.2024.105109

Liu, X., Wang, S., Wang, P., & Wu, D. (2019). Automatic grading of programming assignments: An approach based on formal semantics. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET)* (pp. 126–137). IEEE. https://doi.org/10.1109/ICSE-SEET.2019.00022

Liu, Y., Cao, S., & Chen, G. (2024). Research on the Long-term Mechanism of Using Public Service Platforms in National Smart Education—Based on the Double Reduction Policy. *Sage Open,* 14(1). https://doi.org/10.1177/21582440241239471

Mohammed, P. S., & 'Nell'Watson, E. (2019). Towards inclusive education in the age of artificial intelligence: Perspectives, challenges, and opportunities. *Artificial Intelligence and Inclusive Education: Speculative futures and emerging practices*, 17–37. https://doi.org/10.1007/978-981-13-8161-4_2

Nabil, R., Mohamed, N. E., Mahdy, A., Nader, K., Essam, S., & Eliwa, E. (2021). EvalSeer: An Intelligent Gamified System for Programming Assignments Assessment. In *2021 International Mobile, Intelligent, and Ubiquitous Computing Conference (MIUCC)*, Cairo, Egypt, 2021, pp. 235–242. IEEE. https://doi.org/10.1109/MIUCC52538.2021.9447629

Parihar, S., Dadachanji, Z., Singh, P. K., Das, R., Karkare, A., & Bhattacharya, A. (2017). Automatic grading and feedback using program repair for introductory programming courses. In *Proceedings of the 2017 ACM Conference on Innovation and Technology in Computer Science Education* (pp. 92–97). https://doi.org/10.1145/3059009.3059026

Petropoulou, O., Kasimatis, K., Dimopoulos, I., & Retalis, S. (2014). LAe-R: A new learning analytics tool in Moodle for assessing students' performance. *Bulletin of the IEEE Technical Committee on Learning Technology, 16*(1), 1–13.

Pratama, M. P., Sampelolo, R., & Lura, H. (2023). Revolutionizing education: Harnessing the power of artificial intelligence for personalized learning. *Klasikal: Journal of Education, Language Teaching and Science, 5*(2), 350–357. https://doi.org/10.52208/klasikal.v5i2.877

Ramalingam, V. V., Pandian, A., Chetry, P., & Nigam, H. (2018, April). Automated essay grading using machine learning algorithm. In *Journal of Physics: Conference Series* (Vol. 1000, p. 012030). IOP Publishing. https://doi.org/10.1088/1742-6596/1000/1/012030

Ramesh, D., & Sanampudi, S. K. (2022). An automated essay scoring system: A systematic literature review. *Artificial Intelligence Review, 55*(3), 2495–2527. https://doi.org/10.1007/s10462-021-10068-2

Sirhaug, J. (2018). *Automatic Grading of Programming Exams* (Master's thesis, NTNU). http://hdl.handle.net/11250/2570461. Accessed 23 June 2024.

Stikkolorum, D. R., van der Putten, P., Sperandio, C., & Chaudron, M. (2019). Towards automated grading of UML class diagrams with machine learning. In *Proceedings of the 31st Benelux Conference on Artificial Intelligence (BNAIC 2019) and the 28th Belgian Dutch Conference on Machine Learning (Benelearn, 2019)* (CEUR Workshop Proceedings, Vol. 2491, pp. 1–10). CEUR-WS.org. http://ceur-ws.org/Vol-2491/paper80.pdf. Accessed 23 June 2024.

Tarcsay, B. (2023). *Use of Machine Learning Methods in Automatic Assessment Programming Assignments*. Masters Theses Submitted to School of Enterprise Computing and Digital Transformation, Technological University Dublin, January 2023. https://doi.org/10.21427/EQW1-3S76

Valenti, S., Neri, F., & Cucchiarelli, A. (2003). An overview of current research on automated essay grading. *Journal of Information Technology Education: Research, 2*(1), 319–330. https://doi.org/10.28945/319

Wartman, S. A., & Combs, C. D. (2018). Medical education must move from the information age to the age of artificial intelligence. *Academic Medicine, 93*(8), 1107–1109. https://doi.org/10.1097/ACM.0000000000002044

Wu, L., Long, Y., Gao, C., Wang, Z., & Zhang, Y. (2023). MFIR: Multimodal fusion and inconsistency reasoning for explainable fake news detection. *Information Fusion, 100*, 101944. https://doi.org/10.1016/j.inffus.2023.101944

Xu, T., Gao, Q., Ge, X., & Lu, J. (2024a). The relationship between social media and professional learning from the perspective of pre-service teachers: A survey. *Education and Information Technologies, 29*(2), 2067–2092. https://doi.org/10.1007/s10639-023-11861-y

Xu, W., Xing, Q., Yu, Y., & Zhao, L. (2024b). Exploring the influence of gamified learning on museum visitors' knowledge and career awareness with a mixed research approach. *Humanities and Social Sciences Communications, 11*(1), 1055. https://doi.org/10.1057/s41599-024-03583-4

Zawacki-Richter, O., Marín, V. I., Bond, M., & Gouverneur, F. (2019). Systematic review of research on artificial intelligence applications in higher education–where are the educators? *International Journal of Educational Technology in Higher Education, 16*(1), 1–27. https://doi.org/10.1186/s41239-019-0171-0

Zhai, X., Chu, X., Chai, C. S., Jong, M. S. Y., Istenic, A., Spector, M., ... & Li, Y. (2021). A Review of Artificial Intelligence (AI) in Education from 2010 to 2020. *Complexity*, *2021*(1), 8812542. https://doi.org/10.1155/2021/8812542

Zhang, K., & Aslan, A. B. (2021). AI technologies for education: Recent research & future directions. *Computers and Education: Artificial Intelligence, 2*, 100025. https://doi.org/10.1016/j.caeai.2021.100025

Zhang, J., Yang, D., Li, W., Zhang, H., Li, G.,... Gu, P. (2024). Resilient Output Control of Multiagent Systems With DoS Attacks and Actuator Faults: Fully Distributed Event-Triggered Approach. *IEEE Transactions on Cybernetics,* 1–10. https://doi.org/10.1109/TCYB.2024.3404010

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Springer Nature or its licensor (e.g. a society or other partner) holds exclusive rights to this article under a publishing agreement with the author(s) or other rightsholder(s); author self-archiving of the accepted manuscript version of this article is solely governed by the terms of such publishing agreement and applicable law.

Authors and Affiliations

Ishaya Gambo[1] · Faith-Jane Abegunde[1] · Omobola Gambo[2] · Roseline Oluwaseun Ogundokun[3,4,5] · Akinbowale Natheniel Babatunde[6] · Cheng-Chi Lee[7,8]

✉ Roseline Oluwaseun Ogundokun
ogundokun.roseline@lmu.edu.ng

✉ Cheng-Chi Lee
cclee@mail.fju.edu.tw

[1] Department of Computer Science and Engineering, Obafemi Awolowo University, Ile-Ife 220282, Osun State, Nigeria

[2] Department of Arts and Social Science Education, Lead City University, Ibadan 200005, Nigeria

[3] Department of Software Engineering, Kaunas University of Technology, Kaunas 44249, Lithuania

[4] Department of Computer Science, Landmark University, Omu Aran 251103, Nigeria

[5] Department of Computer Systems Engineering, Tshwane University of Technology (TUT), Pretoria, South Africa

[6] Department of Computer Science, Kwara State University, Malete 241103, Nigeria

[7] Dept. of Library and Information Science, Fu Jen Catholic University, New Taipei City 24206, Taiwan

[8] Dept. of Computer Science and Information Engineering, Asia University, Taichung City 41354, Taiwan