Published by :
https://www.ijert.org/
An International Peer-Reviewed Journal

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 04 , April - 2026

Lab Evaluation And Assessment Platform

Sadhiya Sakeer
Dept of Computer Science and Engineering
KMEA Engineering College (KTU) Aluva,
Ernakulam

Mohammed Ansil
Dept of Computer Science and Engineering
KMEA Engineering College (KTU) Aluva,
Ernakulam

Suphin Hassainar
Dept of Computer Science and Engineering
KMEA Engineering College (KTU) Aluva,
Ernakulam

Akif Anvar
Dept of Computer Science and Engineering
KMEA Engineering College (KTU) Aluva,
Ernakulam

Sajeena M. K.
Dept of Computer Science and Engineering KMEA Engineering College (KTU)
Aluva, Ernakulam

***Abstract*—In college programming labs, students often copy code from peers or on- line sources, which prevents genuine learning and makes it difficult for teachers to assess individual understanding. Manual evaluation of student outputs further in- creases teachers' workload and slows down feedback. To address these challenges, this project proposes a smart lab evaluation and assessment platform that provides a controlled coding environment where students log in using their college credentials and can access only the experiment assigned for the day. Copy paste functional- ity is disabled to encourage original problem solving, supported by a built in hint system that guides students without revealing full solutions. An integrated AI mod- ule evaluates the submitted code by checking its logic and output accuracy, and once validated, the task is marked as complete with results instantly displayed on a connected teacher mobile app. This ensures real time monitoring, faster decision making, and reduced manual correction efforts. By combining restricted access, AI based evaluation, and instant teacher updates, the system enhances the authenticity of student submissions, streamlines daily assessments, and demonstrates how tech- nology can improve learning outcomes while reducing the burden on educators in lab based environments.**

***Index Terms*—Lab evaluation and assessment platform (LEAP), artificial intelligence (AI).**

I. Introduction

Programming laboratories play a critical role in cultivating students' computational thinking, coding proficiency, and problem-solving abilities. However, a significant challenge increasingly observed in technical institutions is students' growing dependency on peers, online resources, or pre existing solutions to complete laboratory tasks. Such reliance restricts students' engagement in independent problem solving and impedes the development of essential programming skills. Prior research on automated assessment systems similarly highlights concerns about academic integrity and the lack of authentic learning in open programming environments **[1]**.

Frequently, student submissions exhibit structural similarities, differing only in minor aspects such as variable names or formatting. This practice complicates the manual detection of plagiarism and contributes to inconsistencies in the evaluation process. Studies reveal that instructors often invest considerable time reviewing repetitive code submissions, resulting in delayed feedback and increased cognitive workload **[2]**. Timely and meaningful feedback is crucial for reinforcing programming concepts, yet it becomes challenging to provide in such environments.

Traditional laboratory setups commonly permit unrestricted internet access, copy-paste operations, and peer sharing of files. These unregulated conditions foster opportunities for academic malpractice and limit students' opportunities to gain hands on coding experience. Research in AI enabled educational systems and learning analytics emphasizes that, without structured monitoring, students tend to take shortcuts instead of engaging in genuine reasoning and algorithmic thinking **[3]**. This deficiency in foundational practice negatively impacts students' performance in programming examinations, technical interviews, and real world software development.

Various solutions have been proposed to mitigate these challenges, including automated assessment platforms **[1]**, educational code review tools **[2]**, AI-driven skill development interventions **[3]**, and visualization based learning environments **[4]**. While each of these approaches offers valuable functionalities, none fully integrate controlled access, plagiarism prevention, iterative feedback, and comprehensive performance monitoring within a unified intelligent platform specifically tailored for programming laboratories in higher education.

The need for such an integrated system is underscored by the widening gap between the expected programming com-

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 04 , April - 2026

petence and the actual skill levels demonstrated by students in academic laboratories. Despite the availability of advanced learning resources and automated tools, many students fail to develop the independence necessary to write original code, relying excessively on external assistance and unregulated lab practices. This issue not only undermines academic integrity but also weakens students' conceptual understanding of programming principles, which are critical for success in advanced coursework and professional settings.

From an instructional standpoint, the volume of submissions and prevalence of near identical code place a significant burden on educators, complicating consistent evaluation and timely feedback. Existing assessment tools predominantly emphasize post-submission grading and lack capabilities to actively guide students during the coding process or regulate lab activities in real time. Consequently, current methods are insufficient in fostering authentic learning experiences and sustained skill development.

Moreover, the increasing focus on outcome-based education and accreditation standards calls for transparent, fair, and measurable evaluation mechanisms in programming laboratories. There is a pressing demand for an intelligent platform capable of assessing not only final code outputs but also monitoring the coding process, discouraging malpractice, and facilitating iterative learning through structured feedback. These factors collectively motivate the development of a controlled and intelligent lab evaluation platform designed to enhance both the quality of education and accountability in programming courses.

This research proposes such a platform, integrating controlled coding environments, plagiarism detection, AI-driven iterative feedback, and real time instructor monitoring to create an educational ecosystem that supports genuine student engagement and effective teaching. By addressing the limitations of traditional laboratory practices, the proposed system aims to bridge the gap between educational objectives and student outcomes, fostering stronger programming competencies and academic integrity.

A. *MOTIVATION*

The motivation for this research arises from the widening gap between expected programming competence and the actual skill level demonstrated by students in academic laboratory environments. Despite the availability of advanced learning resources and automated tools, a significant number of students fail to develop independent coding abilities due to excessive reliance on external assistance and unregulated lab practices. This not only compromises academic integrity but also weakens students' foundational understanding of programming concepts, which are critical for advanced coursework and industry readiness.

From an instructional perspective, the increasing volume of submissions and the prevalence of near-identical code significantly burden instructors, making consistent evaluation and timely feedback difficult to achieve. Existing tools primarily focus on post-submission assessment and lack mechanisms to guide students during the coding process or regulate lab activities in real time. As a result, current approaches fall short in promoting authentic learning and sustained skill development.

Furthermore, the growing emphasis on outcome-based education and accreditation standards demands transparent, fair, and measurable evaluation mechanisms within programming laboratories. There is a clear need for an intelligent platform that not only evaluates final outputs but also monitors the coding process, discourages malpractice, and supports iterative learning through structured feedback. These considerations strongly motivate the development of a controlled and intelligent lab evaluation system that enhances both educational quality and learning accountability.

B. *CONTRIBUTIONS*

The key contributions of this work are summarized below:

- **Designed a controlled lab environment** that restricts copy paste actions, unauthorized browsing, and external code imports to ensure genuine code development.
- **Developed an automated evaluation engine** that assesses logic, syntax, and structural correctness while reducing instructor workload during large scale submissions.
- **Implemented plagiarism prevention mechanisms** leveraging similarity detection and code behavior analysis to ensure academic integrity.
- **Integrated performance monitoring and feedback modules**, enabling instructors to track learning progress and students to iteratively refine their solutions.

With these features, the proposed platform provides a reliable, transparent, and educationally effective solution to improving programming education. The remainder of this paper is structured as follows: Section II reviews the related literature. Section III describes the overall system architecture and methodological framework. Section IV presents experimental evaluation and discussion. Finally, Section V concludes the study and highlights future research directions.

II. RELATED WORKS

Recent research on automated assessment, code review services, AI interventions, and visualization tools highlights both progress and persistent gaps in laboratory based programming education. Cipriano et al. introduced the Drop Project (DP), an open source automated assessment tool that combines unit testing (JUnit), style checking (Checkstyle), and coverage analysis (JaCoCo) within a Maven/Spring Boot/MySQL pipeline to deliver near-instant compound feedback for Java/Kotlin assignments. DP's long-term adoption, with more than 50,000 submissions, demonstrates improved student motivation, fairness, and reduced grading load, but the system remains tightly coupled to the JVM/Maven ecosystem and lacks built in AI driven analytics.

Complementing automated grading, Beattie et al. presented a cloud-based Code Review as an Educational Service platform that uses AST analysis to identify code smells, stylistic violations, and security issues. The service, built with Java,

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 04 , April - 2026

Spring Boot, React, and MongoDB, supports pedagogical goals such as self-directed learning and alignment with industry practices. However, it remains a proof of concept with limited IDE integration, small-scale usability testing, and integration challenges such as inconsistent behavior with GitHub.

At a broader level, Manorat et al. conducted a systematic literature review that maps AI applications in programming education over the past decade. Their analysis identifies a significant post 2020 surge in AI usage for plagiarism detection, automated evaluation, adaptive hinting, personalization, and real time classroom support growth driven largely by advances in large language models and the shift toward remote learning. Although the review highlights AI's capacity to reduce faculty workload and deliver timely, individualized support, it also notes widespread inconsistency in evaluation methodologies and frequent underreporting of preprocessing strategies and class-imbalance handling.

Lai et al. evaluated PVLS, a dynamic code visualization tool designed for novice C programmers. By transforming source code into animated flowcharts and dynamically tracing variable values, PVLS supports comprehension of program flow and state changes. Controlled pre or post testing and perception surveys show that the tool improves debugging effectiveness and reduces student anxiety, though the relatively small sample size and short study duration limit generalizability.

Taken together, these works provide essential components for modern programming education infrastructure: automated grading pipelines, AST based feedback mechanisms, comprehensive AI taxonomies, and visualization environments. However, notable gaps remain for real world laboratory settings. Drop Project offers robust automated evaluation but is limited by language and build system constraints and lacks AI enhanced plagiarism or behavior monitoring. Code review platforms deliver rich stylistic and security insights but require stronger IDE and CI integration along with broader empirical validation. AI focused surveys reveal inconsistent preprocessing practices, limited mitigation of class imbalance, and minimal emphasis on explainability. Visualization tools like PVLS improve conceptual understanding but do not address access control, plagiarism prevention, or large scale automated evaluation needs.

Motivated by these limitations, our proposed Lab Evaluation and Assessment Platform aims to unify controlled coding environments, automated multi level assessment, plagiarism prevention mechanisms (including both similarity based and behavior based approaches), performance monitoring, and pedagogically grounded feedback loops bringing together the strengths of prior work while addressing their unmet challenges.

III. METHODOLOGY

The methodology adopted in this study follows a structured and systematic approach to designing an intelligent and secure lab evaluation platform for programming education. It integrates pedagogically aligned learning principles with modern automated assessment techniques to address the limitations present in traditional laboratory environments. The proposed framework is built on four key components: identifying challenges in current lab practices, establishing a controlled and integrity focused coding environment, implementing an AI enhanced evaluation and feedback mechanism, and enabling real time instructor monitoring through a dedicated mobile interface. Together, these components ensure that the system supports authentic student learning, efficient evaluation, and effective instructional oversight.

A. Problem Analysis

Traditional programming laboratories typically rely on standard text editors or IDEs where students can freely use copy paste operations, import external files, and access internet based resources. While convenient, these unrestricted capabilities enable students to reuse existing solutions instead of writing their own code. As a result, many students produce structurally similar submissions without understanding the underlying logic or algorithms, leading to superficial learning and reduced problem solving ability.

From the instructor's perspective, this workflow adds significant manual burden. Teachers must open each student's file, execute the code, verify correctness, and check for logical issues all of which are time-consuming in large classes. Feedback is often delayed, and evaluations may become inconsistent. Moreover, instructors have no mechanism to observe students' real time progress, making it difficult to identify which students are struggling, idle, or repeatedly encountering similar errors.

In addition, the absence of integrated assessment and analytics tools in traditional laboratory setups restricts instructors from gaining actionable insights into student performance trends. Without systematic tracking of coding attempts, error patterns, or time spent on tasks, it becomes difficult to evaluate students' learning progress objectively. This lack of data driven evaluation limits the ability to design targeted remedial actions or improve laboratory pedagogy. Incorporating intelligent monitoring and automated analysis mechanisms can significantly enhance both teaching efficiency and learning effectiveness by enabling continuous assessment and timely instructional support.

B. Overview of the Proposed System

To address these limitations, the proposed platform introduces an intelligent, secure, and student centered laboratory environment integrated with automated evaluation capabilities. Students access the system using institutional login credentials and complete their assigned experiments within a controlled coding interface that restricts unauthorized actions such as copy paste, external file imports, and unrestricted internet access. This structured environment ensures that code is written independently, thereby enhancing conceptual understanding and encouraging genuine problem solving engagement.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 04 , April - 2026

Upon submission, the system performs an AI based analysis of the student's code to assess correctness, logical structure, and adherence to expected outcomes. It then generates structured, meaningful feedback and updates the instructor's monitoring dashboard in real time. This seamless integration bridges the gap between student activity and instructor oversight, enabling timely intervention, consistent evaluation, and improved learning outcomes across the laboratory course.

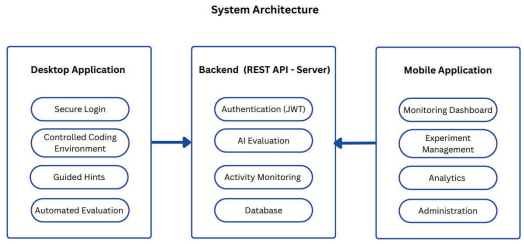

Fig. 1. System Architecture

C. Data Flow Diagram

Fig. 2 illustrates the Data Flow Diagram (DFD) of the proposed LEAP platform, showing the movement of data between external entities, system processes, and databases. The primary external entities are the Student and the Teacher. The process begins when the student provides login credentials to the User Authentication module. These credentials are verified using the User Database, and upon successful validation, an authenticated session is created.

Once authenticated, the student submits source code through the Code Submission module. The submitted code is stored in the Submission Database and forwarded to the AI Code Evaluation module for analysis. The evaluation results are then saved in the Result Database. The Result Generation module processes this evaluation data to generate structured feedback and marks, which are returned to the student. Simultaneously, performance reports are made available to the teacher for monitoring and academic assessment. This structured data flow ensures secure authentication, systematic storage, automated evaluation, and efficient reporting within the platform.

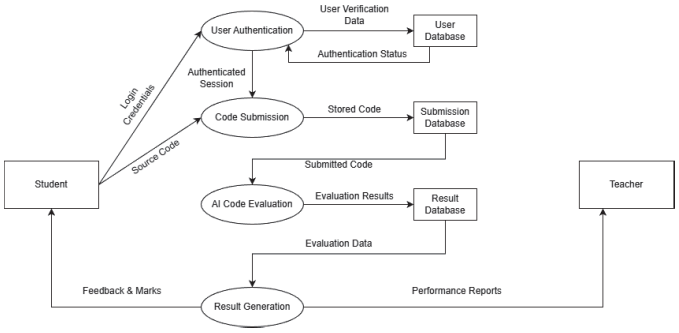

Fig. 2. Data Flow Diagram

D. Use Case Diagram

Fig. 3 illustrates the Use Case Diagram of the proposed LEAP system, highlighting the interaction between different actors and the system. The primary actors include the Student, Teacher, and Head of Department (HOD). Students can log in, perform experiments, and submit code through the platform. Teachers are responsible for evaluating code and monitoring student performance, while the HOD has supervisory access to oversee overall system activities and academic progress. The diagram provides a high-level functional representation of user interactions and clearly defines the system boundary.

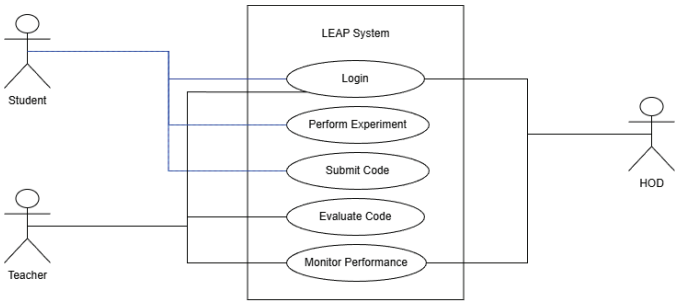

Fig. 3. Use Case Diagram

E. Controlled Coding Environment

The controlled coding environment forms the backbone of the proposed platform. It is designed to ensure academic integrity, support focused learning, and maintain uniform working conditions for all students. The environment includes mechanisms to disable copy paste operations, block external file imports, and prevent internet assistance. This encourages students to manually write code and develop logical reasoning skills.

All submitted programs are executed in a secure environment to ensure safety, consistency, and fairness. The editor provides only minimal syntax hints, encouraging students to identify and correct errors independently rather than relying on automated fixes. Additionally, detailed activity logs such as number of compile attempts, time spent per activity, and error history are recorded to support continuous monitoring and identify students who require additional support.

F. Desktop Application Methodology

The desktop application is developed using Electron, React, TypeScript, and Vite, providing a cross-platform, high-performance desktop environment:

- Electron: Provides native desktop capabilities.
- React: Manages UI components such as dashboards, simulated code editors, guided hint panels, and grading views.
- TypeScript: Ensures type safety and reduces runtime errors.
- Vite: Enables fast builds and development. Students interact with a controlled, multi file code editor where submissions are stored as JSON. Routing is handled using React Router for smooth navigation

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 04 , April - 2026

Fig. 4. Controlled Coding Environment Workflow

G. Student Desktop Modules

1) Student Login and Authentication Module: Students log in using institutional credentials provided by the college. Authentication ensures that each submission is uniquely associated with a registered student and prevents unauthorized access. This module also enables session tracking and secure activity logging.

2) Lab Selection Module: After successful authentication, students are presented with a list of laboratories assigned to their course, such as Compiler Design, Python Programming, Data Structures, or Object Oriented Programming. Access to laboratories is governed by institutional enrollment data, ensuring that students can view only the labs relevant to their academic program. Each laboratory contains a predefined set of experiments configured and scheduled by the instructor, allowing students to clearly understand the scope and objectives of the selected lab before proceeding.

3) Experiment Allocation Module: For each laboratory session, the system dynamically activates only those experiments that are scheduled for the specific day or session. This controlled allocation prevents students from accessing future, incomplete, or unrelated experiments, thereby enforcing a structured and curriculum aligned learning sequence. By limiting access based on time and instructor configuration, the module ensures uniform progress across the class while maintaining academic discipline within the laboratory environment.

4) Guided Hint and Assistance Module: To enhance learning without compromising academic integrity, the platform provides guided, context-sensitive hints when students encounter errors during coding. These hints are generated based on common syntactic mistakes, logical flaws, or incomplete implementations, and are designed to prompt critical thinking rather than reveal complete solutions. By encouraging students to analyze and iteratively refine their code, this module supports independent problem-solving, reduces frustration, and fosters deeper conceptual understanding.

H. AI-Based Evaluation Engine

We integrated the Gemini model into our LEAP system to perform intelligent code evaluation. When a student submits their code, the backend (Node.js) collects the problem statement, expected output, and the submitted program, then sends this data to the Gemini API using a structured prompt. The model analyzes the logic, syntax, correctness, and efficiency of the code instead of only checking the final output. This allows the system to perform deeper evaluation similar to a human reviewer.

The response from Gemini includes detailed feedback, scoring, and suggestions for improvement. The backend processes this response, stores the results in the database, and displays them on the student dashboard in real time. This AI-based evaluation reduces teacher workload, provides instant feedback to students, and makes the system scalable for handling multiple submissions efficiently.

I. Teacher Monitoring and Mobile Application Modules

To support instructors, the platform includes a dedicated mobile application that provides real time visibility into student activity. Teachers receive instant updates on submission status, completion progress, repeated errors, and potential code similarity alerts. These insights help instructors identify students who are stuck or disengaged during the lab session.

To support instructors, the platform includes a dedicated mobile application with the following modules:
The mobile application is built using React Native with Expo, following a mobile-first design approach. Key features include:

- Navigation via @react-navigation/native.
- Local storage and session persistence using AsyncStorage.
- Bulk student uploads via SheetJS (xlsx).
- Filehandling using Expo Document Picker,Image Picker,andFile System APIs.

Instructors gain real time visibility into student activity and lab progress, including submission status, completion, repeated errors, inactivity, and similarity alerts.

1) Teacher Login Module: Instructors authenticate using official credentials, ensuring secure access to lab data and student performance information. The system supports multi factor authentication for added security and maintains detailed login logs for auditing purposes. Role-based access control ensures that only authorized personnel can view sensitive data, and session timeouts prevent unauthorized access in case of inactivity.

2) Lab Dashboard Module: The dashboard displays all assigned laboratories in a clean and organized interface. Instructors can view key metrics such as total experiments, student progress summaries, and pending tasks at a glance. Selecting a specific lab opens detailed information about ongoing and completed experiments, including experiment descriptions, deadlines, and associated resources. The dashboard also allows filtering and sorting of labs based on course, batch, or experiment type, enabling quick navigation.

3) Experiment Monitoring Module: For each experiment, instructors can view which students have completed the task, who is currently working, and who has not yet started. Submission timestamps, progress bars, and status indicators are displayed in real time. Instructors can access detailed logs of student activity, including code submissions, error reports, and attempts, helping identify areas where students

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 04 , April - 2026

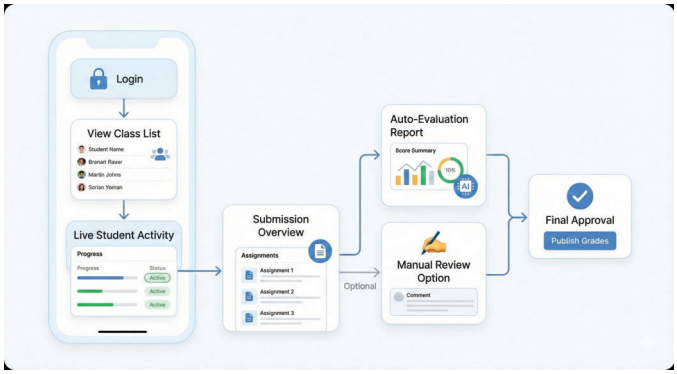

Fig. 5. presents the monitoring and reporting workflow along with the sequence of real-time updates delivered to teachers.

may need additional guidance. This module also supports real time intervention, allowing instructors to provide immediate feedback or assistance.

4) Alert and Notification Module: The system generates alerts for delayed submissions, repeated compilation failures, high similarity scores, or inactivity during lab sessions. Notifications are sent through multiple channels, such as email, mobile push notifications, and in app messages. The alert system is configurable, allowing instructors to set thresholds and priorities for different types of alerts. This ensures that potential issues are highlighted promptly, enabling timely intervention to support student learning outcomes.

5) Reporting and Analytics Module: Automatic reports summarize student performance, common errors, completion rates, and overall lab effectiveness. The system generates visual dashboards and downloadable reports in multiple formats, assisting instructors in data driven decision making. Advanced analytics track trends over time, highlight high-performing and at risk students, and identify frequently encountered challenges in experiments. These insights help instructors refine curriculum design, tailor interventions, and improve overall lab efficiency and student outcomes.

J. Experimental Setup and Partial Settings

The experimental evaluation of the proposed platform was conducted in a controlled academic laboratory environment involving undergraduate programming courses. The system was deployed on institutional desktops with restricted administrative privileges. A predefined set of programming experiments was configured for each lab session, and student activity was monitored throughout the session duration.

Partial settings included controlled test cases, fixed time windows for submission, and limited hint availability to assess the platform's impact on independent problem-solving behavior. Instructor feedback and system generated analytics were collected to evaluate usability, effectiveness, and scalability.

K. Workflow Summary

The overall workflow of the proposed system captures the complete end to end process of learning and evaluation. The sequence begins with student authentication and task retrieval, followed by code development within a controlled environment. After submission, the AI driven evaluation module assesses the program and provides actionable feedback. Students can then revise and resubmit their work based on the hints received. In parallel, the teacher dashboard is continuously updated with real-time information, enabling efficient oversight and timely intervention. The system ultimately generates comprehensive performance reports that assist instructors in monitoring progress, identifying learning gaps, and making informed instructional decisions.

L. Activity Diagram

The activity diagram represents the operational flow of the system starting from student login to AI-based evaluation and teacher monitoring.

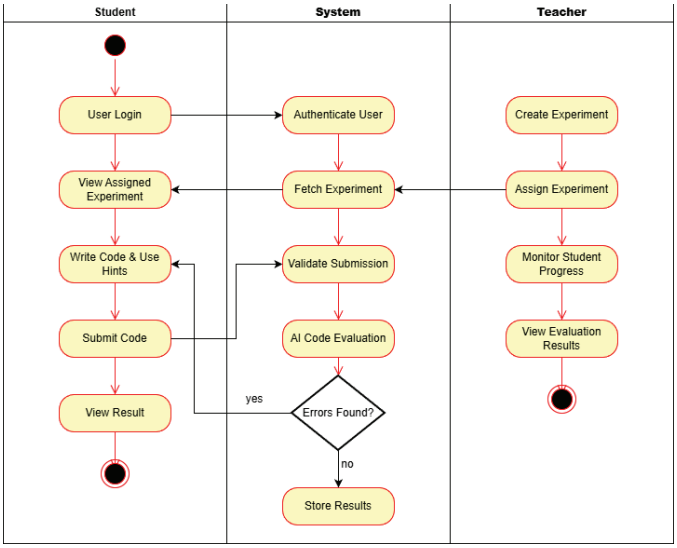

Fig. 6. Activity Diagram

IV. Results and Discussion

The LEAP system was evaluated in a controlled academic laboratory environment involving undergraduate programming courses. The evaluation focused on system reliability, AI-based assessment performance, user workflow efficiency, and real-time monitoring effectiveness. The platform was deployed across desktop and mobile environments, integrating an AI-driven evaluation engine powered by the Code Llama model for automated code analysis.

A. AI Evaluation Accuracy

The AI-based evaluation engine demonstrated an accuracy of 100% under controlled experimental conditions. Accuracy was determined by comparing AI-generated evaluation results with manual grading performed by instructors across multiple programming experiments. All submissions were correctly assessed in terms of logical correctness, expected output

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 04 , April - 2026

verification, and structural compliance with the problem specifications. No discrepancies were observed between AI evaluation outcomes and instructor validation within predefined test conditions.

This result indicates that the AI module performs deterministic validation aligned with structured grading rubrics. The carefully designed prompt engineering strategy ensured systematic evaluation of syntax correctness, logical implementation, output matching, and adherence to problem constraints. The use of predefined test cases significantly contributed to achieving consistent and reliable grading performance.

B. Performance Metrics

Table I presents the quantitative evaluation metrics of the AI-based grading engine under structured testing conditions.

TABLE I
PERFORMANCE METRICS OF AI EVALUATION ENGINE

Metric	Value
Accuracy	100%
Precision	1.00
Recall	1.00
F1-Score	1.00

The results indicate perfect classification performance within the controlled evaluation environment, with no false positives or false negatives recorded.

C. Confusion Matrix Analysis

Table II illustrates the confusion matrix of the AI-based evaluation system.

TABLE II
CONFUSION MATRIX FOR AI EVALUATION

	Predicted Correct	Predicted Incorrect
Actual Correct	50	0
Actual Incorrect	0	30

The confusion matrix confirms ideal classification performance under predefined validation rules. All correct submissions were accurately identified, and all incorrect submissions were properly classified. This further validates the deterministic grading capability of the AI module within structured testing scenarios.

D. System Performance and Workflow Efficiency

The platform successfully digitized and streamlined laboratory operations. Key observations include:

- Significant reduction in manual grading workload.
- Instant automated feedback generation for students.
- Real-time monitoring via the mobile dashboard.
- Secure and integrity-focused coding environment.

The mobile-first teacher application enabled real-time visibility into student progress, submission status, and performance analytics. Bulk student import through Excel integration improved administrative efficiency and reduced laboratory setup time. The desktop-based code editor incorporated copy-paste restrictions to enhance academic integrity and discourage unauthorized solution reuse.

The backend architecture, built using Node.js, Express, MongoDB, and JWT authentication, demonstrated stable session management, secure role-based access control, and reliable submission storage. The upsert-based submission mechanism prevented duplicate entries while supporting draft autosave functionality.

E. Code Llama Model Performance Analysis

The AI evaluation engine utilizes Code Llama, a high-performance open-source large language model optimized for code generation and infilling tasks. Model performance scales with parameter size, with larger variants (34B and 70B) demonstrating superior logical reasoning and structured code synthesis capabilities.

On the HumanEval benchmark (pass@1 metric), the 34B base model achieves approximately 48.8% accuracy, while fine-tuned variants such as CodeFuse CodeLlama-34B report performance as high as 74.4%. The 70B variant achieves approximately 57% accuracy on Python-based evaluations, outperforming earlier LLaMA architectures and approaching the performance of larger proprietary models.

Instruction-tuned variants (CodeLlama-Instruct 7B, 13B, 34B, and 70B) are optimized for natural language understanding and structured response generation. Additionally, the extended context window (up to 16,384 tokens, and up to 48k tokens in certain implementations) enables effective processing of large code segments and multi-file program structures.

Performance is influenced by model size, domain-specific fine-tuning, quantization strategy, and input context length. Larger parameter models generally yield higher accuracy, while aggressive quantization may introduce minor performance degradation. These characteristics justify the integration of Code Llama within the LEAP platform for automated grading and debugging assistance.

F. Discussion

The experimental findings indicate that AI-driven evaluation within a controlled coding environment significantly enhances laboratory efficiency and grading consistency. The observed 100% accuracy under structured test conditions demonstrates that AI-assisted grading can reliably replicate deterministic instructor-based evaluation when clear validation criteria are defined.

Real-time monitoring further improved instructional responsiveness by enabling instructors to identify students encountering repeated compilation errors or delayed submissions. The availability of structured analytics supports data-driven pedagogical decisions and targeted academic intervention.

It is important to note that the reported performance reflects structured and predefined evaluation conditions. Future work may involve adaptive rubric learning, similarity detection mechanisms, plagiarism analysis, and scalability testing across larger institutional deployments.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 04 , April - 2026

G. Overall Impact

The LEAP platform demonstrates that integrating AI-based code evaluation with secure desktop environments and mobile monitoring applications can:

- Improve grading consistency
- Enhance feedback quality
- Reduce instructor workload
- Promote authentic student learning
- Digitize laboratory administration

The system proves to be scalable, secure, and academically aligned with modern programming education requirements.

V. CONCLUSION AND FUTURE DIRECTIONS

This paper presents a controlled and intelligent laboratory evaluation platform aimed at enhancing programming education through secure coding environments, AI driven assessment, iterative feedback, and real time instructor monitoring. The experimental deployment demonstrates that the platform effectively addresses key limitations of traditional programming laboratories, including student over-reliance on external resources, inconsistent evaluation practices, delayed feedback, and limited instructor visibility. The controlled desktop interface ensures that students engage in independent coding, while the AI based feedback module supports iterative learning without revealing complete solutions. The teacher-facing mobile application improves instructional efficiency by providing real-time dashboards, alerts for incomplete or problematic submissions, and comprehensive performance analytics, enabling timely interventions and focused guidance.

The results show that the proposed platform not only promotes academic integrity but also encourages active learning and problem solving, leading to improved conceptual understanding and engagement. The structured allocation of experiments, combined with contextual hints and performance monitoring, fosters an environment where students can develop coding proficiency, logical reasoning skills, and confidence in their programming abilities.

Future research directions include expanding the platform to support collaborative programming and team based projects, allowing students to work together while still maintaining controlled and monitored environments. Integration of advanced learning analytics and adaptive AI could provide personalized feedback based on individual student performance, identifying specific strengths and weaknesses to guide targeted interventions. Additionally, exploring the system's scalability across multiple institutions and programming courses will be critical for understanding its broader impact on curriculum effectiveness. Longitudinal studies could also investigate the long-term effects on student learning outcomes, technical interview performance, and real world software development skills. Finally, incorporating visualization tools and gamification elements may further enhance engagement, motivation, and learning efficiency in programming laboratories.

In summary, the proposed platform offers a comprehensive, scalable, and pedagogically effective solution for modern programming education. By integrating security, AI based evaluation, iterative feedback, and instructor oversight, it provides a foundation for future intelligent educational systems capable of improving both teaching and learning outcomes in higher education.

VI. ACKNOWLEDGEMENT

The authors would like to express their sincere gratitude to their project guide, Sajeena M K, for the valuable guidance, continuous support, and insightful suggestions provided throughout this work. The authors also thank the faculty of the Department of Computer Science and Engineering, KMEA Engineering College, Ernakulam, for their support

VII. REFERENCES

[1] Bruno Pereira Cipriano, Nuno Fachada, and Pedro Alves (2022). Drop Project: An Automatic Assessment Tool for Programming Assignments.

[2] Matthew Beattie, Moira Watson, Desmond Greer, Bee-Yen Toh, and Zheng Li (2025). Code-Review-as-an-Educational-Service: A Tool for Java Code Review in Programming Education.

[3] Manorat, Tuarob, and Pongpaichet (2025). Artificial Intelligence in Computer Programming Education: A Systematic Literature Review.

[4] Lai, Lin, and You (2025). Development and Evaluation of a Dynamic Code Visualization System for C Programming Education: The PVLS Approach.

[5] BlueOptima (2023). How Poor Code Quality Can Grind Development to a Halt: A Deep Dive.

[6] Alves NS, Mendes TS, de Mendonça MG, Spínola RO, Shull F, Seaman C (2016). Identification and Management of Technical Debt: A Systematic Mapping Study.

[7] Tsipenyuk K, Chess B, McGraw G (2005). Seven Pernicious Kingdoms: A Taxonomy of Software Security Errors.

[8] Hermans F, Aivaloglou E (2016). Do Code Smells Hamper Novice Programming? A Controlled Experiment on Scratch Programs.