iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 14 **Issue:** V **Month of publication:** May 2026

DOI: https://doi.org/10.22214/ijraset.2026.83019

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 14 Issue V May 2026- Available at www.ijraset.com

ExamNova: Academic Tracking with Integrated Code Evaluation LMS

C. U. Chauhan[1], Shweta K. Dhote[2], Trupti Y. Balbudhe[3], Sanika L. Bodhale[4], Tushar S. Datkar[5], Pranav C. Gore[6]
Department of Computer Science and Engineering, Government College of Engineering, Chandrapur, Maharashtra, India

Abstract: ***Contemporary Learning Management Systems (LMS), notably Moodle, provide robust academic content delivery and engagement tracking but remain fundamentally deficient in one critical dimension: the assessment and analytics of practical coding skills. Existing analytics plugins such as IntelliBoard aggregate engagement data from course activities yet possess no mechanism to capture, evaluate, or visualize coding-specific performance metrics. This gap deprives educators of insight into how students write, debug, and iteratively improve code—competencies that are foundational to Computer Science and Engineering education. The present work introduces ExamNova, a dual-component system built atop Moodle LMS. The first component, CodeJudge, is a custom Moodle local plugin (local_codejudge) developed from the ground up. It embeds a feature-rich in-browser code editor based on the Ace editor library—supporting C, C++, Java, and Python—and routes student submissions to a secure Python Flask microservice. Code execution is sandboxed within Docker containers configured with strict resource limits: no network access, 128 MB memory cap, 0.5 CPU share, and a process limit, ensuring both security and reproducibility. The second component enhances the IntelliBoard analytics plugin by extending its data collection to CodeJudge tables via SQL, enabling dedicated coding analytics dashboards including per-student performance trends, language-wise usage statistics, question difficulty analysis, at-risk student identification, and competitive leaderboards. The integrated system is validated through functional testing, boundary-condition evaluation, and Docker security assessment, confirming accurate test-case evaluation, secure execution, and meaningful analytics generation.***

Keywords: ***Moodle LMS, Automated Code Evaluation, Docker Sandboxing, Learning Analytics, IntelliBoard, Online Judge, Coding Assessment, PHP Plugin, Python Flask, Ace Editor.***

I. INTRODUCTION

Digital education platforms have transformed how academic institutions design, deliver, and assess learning. Moodle (Modular Object-Oriented Dynamic Learning Environment), the most widely deployed open-source LMS globally, offers a comprehensive environment for content management, assignment submission, quiz administration, and grading. Its extensible plugin architecture makes it highly adaptable—yet it was never designed with programming assessment in mind, a limitation that grows increasingly significant as Computer Science and Engineering programs expand online. In engineering education, hands-on coding proficiency is a non-negotiable graduate attribute. While Moodle accepts code file submissions, it cannot compile, execute, or evaluate correctness through test cases, nor capture granular behavioral data such as number of attempts, time per attempt, or specific errors encountered. Instructors are thus forced to rely on external judges like HackerRank or LeetCode with no LMS integration, or resort to manual evaluation—neither of which scales or provides the rich, integrated data stream needed to detect struggling students early, flag problematic questions, or track coding improvement across a semester.

ExamNova addresses this gap by embedding a Docker-sandboxed multi-language code judge directly within Moodle, coupled with a coding-aware analytics layer through IntelliBoard. This paper presents the complete architecture, implementation methodology, testing framework, and educational implications of the system.

A. Problem Statement

Moodle, despite its widespread adoption in engineering education, does not natively support in-browser code editing, multi-language compilation, Docker-based secure execution, or test-case-driven evaluation of student programs. The IntelliBoard analytics plugin, while providing valuable engagement analytics over general Moodle activities, offers no mechanism to ingest, process, or visualize data specific to coding activity. As a result, instructors in programming-intensive courses are unable to monitor coding performance trends, identify at-risk students based on coding metrics, or perform question-level difficulty analysis within their LMS environment. The absence of an integrated coding assessment and analytics pipeline within Moodle constitutes the central problem that ExamNova seeks to address.

B. Research Objectives

This research pursues the following primary objectives:

1) To design and implement CodeJudge—a custom Moodle local plugin providing in-browser code editing and test-case-based evaluation for C, C++, Java, and Python programs.
2) To develop a secure, Docker-based Python Flask microservice (judge service) executing student code in isolated containers with enforced resource limits (--network none, --memory 128m, --cpus 0.5, --pids-limit 64, --read-only).
3) To persist all coding activity data—submissions, attempts, and test case results—in five structured database tables integrated with the Moodle schema.
4) To emit Moodle-standard events (submission_created, code_executed, test_case_passed, test_case_failed) ensuring all coding activity is captured in the Moodle logstore.
5) To extend the IntelliBoard analytics plugin to read CodeJudge data and render six dedicated coding analytics dashboards for instructors and administrators.
6) To validate the system through comprehensive functional testing, boundary-condition evaluation, and Docker sandbox security assessment.

C. Scope of Work

The scope of this research encompasses the complete lifecycle of a coding assessment within the Moodle LMS environment: question creation by instructors, test case management, in-browser code editing and submission by students, secure multi-language code execution, result evaluation, data persistence, event emission, and analytics generation. The system is developed as a Moodle local plugin (local_codejudge) and a companion enhancement to IntelliBoard. The judge microservice is implemented as a Flask application containerized with Docker. This work does not encompass a standalone web application, mobile application, or replacement for Moodle's core academic modules.

II. LITERATURE REVIEW

The literature informing ExamNova spans four thematic areas: time-based engagement metrics, analytics methodology, container-based security, and browser-based code editing.

On engagement analytics, Daza et al. (2021) established that time spent on a resource is a stronger predictor of student comprehension than access frequency. A follow-up study (Daza & Traver, 2022) identified inconsistencies in how IntelliBoard and Edwiser Reports calculate active time, reinforcing the need for transparent, well-documented data collection—a principle central to the CodeJudge–IntelliBoard integration. Bakharia and Dawson (2020) further showed that custom SQL queries and third-party plugins unlock richer institutional insights beyond Moodle's default reports, justifying ExamNova's use of custom XMLDB-defined tables. Verbert et al. (2018), in a synthesis of over fifty studies, confirmed that learning analytics tools improve teaching strategies and academic outcomes, yet noted that most existing tools focus on passive content engagement rather than active problem-solving—the precise gap ExamNova addresses. Chang et al. (2017) demonstrated that classification and clustering algorithms applied to LMS log data can yield actionable predictive models of student success, directly informing ExamNova's at-risk detection feature.

On the infrastructure side, Tsai et al. (2019) showed that Docker-based sandboxing with enforced resource limits achieves security comparable to full virtual machine isolation at significantly lower overhead, underpinning the CodeJudge execution architecture. Raja and Bhanu (2020) proposed a Moodle module for Java code evaluation but omitted container-based isolation, leaving risks such as fork bombs, disk exhaustion, and network exfiltration unmitigated—all of which ExamNova explicitly addresses. Finally, Meftah et al. (2021) found that syntax highlighting and inline error indication in browser-based editors measurably reduce novice programmer frustration and improve task completion rates, motivating the adoption of the Ace editor in CodeJudge.

III. SYSTEM DESIGN AND METHODOLOGY

ExamNova is designed as a three-tier architecture operating entirely within the Moodle ecosystem: the Presentation Tier, the Application and Judge Tier, and the Data and Analytics Tier. These tiers interact through well-defined interfaces, ensuring modularity, separation of concerns, and extensibility for future enhancements. The architectural philosophy prioritizes security, reproducibility, and pedagogical utility in equal measure.

A. System Architecture

ExamNova follows a three-tier architecture: a Presentation Tier (Ace editor + teacher dashboards), an Application and Judge Tier (PHP backend + Flask microservice), and a Data and Analytics Tier (five custom DB tables + IntelliBoard dashboards).

B. CodeJudge Plugin Design

The CodeJudge plugin follows Moodle's standard local plugin architecture. Key PHP components include submit.php (AJAX submission handler), index.php (question listing), add_question.php (question and test case creation), view.php (student code editor view), and lib.php (Moodle hook registrations). The Ace editor is loaded as a RequireJS AMD module (amd/src/editor.js) and automatically configured to the selected language mode when a student selects a programming language from the dropdown interface. Results are dynamically rendered into the DOM from the JSON response without requiring a page reload, providing an immediate and responsive user experience.

The Flask judge microservice exposes a POST /run endpoint that compiles (if needed), executes code inside Docker containers, evaluates output against expected results, and returns verdicts with execution time and memory usage.

C. Database Schema

Five custom tables are created via Moodle's XMLDB install.xml, ensuring compatibility with MariaDB, MySQL, and PostgreSQL through Moodle's database-agnostic DML API. All tables use the InnoDB storage engine for ACID transaction support and foreign key enforcement. Text-heavy fields (code, input, expected_output, actual_output) employ LONGTEXT to accommodate programs of arbitrary length. The five custom tables are: codejudge_questions, codejudge_testcases, codejudge_submissions, codejudge_attempts, and codejudge_testcase_results.

D. IntelliBoard Enhancement

A new module, codejudge_datasource.php, implements IntelliBoard's datasource interface and exposes six SQL query methods that power the dedicated analytics dashboards. Six SQL-powered dashboards are exposed: per_student_performance, attempts_vs_success, time_per_question, language_distribution, difficulty_analysis, and at_risk_detection.

E. End-to-End Workflow

The end-to-end ExamNova workflow proceeds through nine sequential stages: (1) The student writes and submits code in the Moodle Ace editor. (2) The CodeJudge plugin validates the submission and retrieves the associated test cases. (3) Code is dispatched to the Flask judge service via cURL. (4) The judge executes code inside Docker containers for each test case and returns structured results. (5) Results—pass/fail verdicts, actual output, execution time, and memory usage—are persisted in the custom database tables. (6) Moodle events are emitted and logged to logstore_standard_log. (7) IntelliBoard reads both the event logs and CodeJudge tables. (8) Analytics metrics are computed, including pass rates, attempt patterns, and time distributions. (9) Dashboards are rendered for teachers, administrators, and students.

IV. TESTING AND RESULTS

Validation of ExamNova was conducted across three sequential testing phases: unit testing of individual PHP classes and the Python runner module; integration testing of communication between the Moodle plugin and the judge service and between the judge service and Docker containers; and system testing of the complete end-to-end workflow from student code submission to instructor analytics visualization. All test cases passed successfully across all three phases.

A. Performance Evaluation

Median round-trip times were: Python ~1.2s, C/C++ ~1.1s, and Java ~2.8s (due to JVM startup overhead). All results fell well within the 5-second timeout threshold, and TLE verdicts were issued within 100ms of the boundary, confirming timing precision. The system is practical for classroom deployment with immediate, actionable feedback.

B. Analytics Validation

The IntelliBoard coding analytics dashboards were validated by inserting a controlled set of synthetic submissions via Moodle's Behat testing framework and verifying that all six dashboard widgets displayed the expected metrics with complete accuracy. The Attempts vs.

Success Rate chart correctly grouped students into attempt bands (1 attempt, 2–3 attempts, 4–5 attempts, and more than 5 attempts) and computed the success rate for each band. The Language-wise Analysis chart correctly tallied submissions per programming language. The Question Difficulty Analysis chart correctly computed pass rates per question grouped by Easy, Medium, and Hard categories. The at-risk detection mechanism correctly flagged students with more than five failed attempts on any question without achieving a passing score, demonstrating actionable early-intervention capability for instructors.

C. Docker Security Assessment

The Docker sandbox was evaluated systematically against common attack vectors to confirm the robustness of the security configuration. Fork bomb attempts were successfully contained by the --pids-limit 64 flag, preventing uncontrolled process proliferation. Network exfiltration attempts were blocked comprehensively by the --network none configuration. Memory exhaustion attempts were capped by --memory 128m with automatic OOM (Out-Of-Memory) kill enforcement. Filesystem write attempts outside the designated /tmp directory were blocked by the --read-only flag combined with --tmpfs /tmp:size=64m. All containers were automatically removed after execution via the --rm flag, preventing resource leakage across successive submissions. These results confirm that the Docker security configuration provides robust isolation of student code execution without requiring dedicated virtual machines or significantly elevated infrastructure costs.

V. RESULTS AND DISCUSSION

A. Before and After Integration: A Comparative Analysis

Prior to ExamNova, Moodle analytics were limited to submission timestamps, page views, quiz scores, and login frequency—recording only final grades with no insight into how students engaged with code. Language preferences, debugging time, and question difficulty were entirely invisible to instructors.

After integration, every attempt, test case verdict, execution time, and language choice is recorded. The six IntelliBoard dashboards allow instructors to distinguish students struggling algorithmically (high attempts, low pass rates) from those disengaged from course content—two distinct profiles requiring different interventions—before the midterm examination.

B. Key Findings

1) Docker sandboxing provides robust, VM-free isolation for secure multi-tenant code evaluation on standard server infrastructure.
2) Moodle event system serves as a decoupled integration bus between CodeJudge and IntelliBoard, preserving maintainability and independent upgradability.
3) Python and Java exhibit higher execution latency than C/C++ due to interpreter and JVM startup overhead, requiring per-language time limit adjustments.
4) At-risk detection based on attempt thresholds effectively identifies struggling students, validated by correlation with examination performance.
5) Coding-specific analytics surface actionable early signals that general engagement metrics cannot detect, enabling timely pedagogical intervention.

C. Challenges and Limitations

Despite the system's validated effectiveness, several challenges merit acknowledgment. The dependency on correct Moodle configuration and plugin compatibility requires institutions to ensure that the CodeJudge plugin and IntelliBoard versions are correctly integrated and maintained in tandem. Server resource consumption represents a practical concern at scale: large numbers of concurrent Docker container instantiations may require infrastructure optimization and load management strategies. Data privacy and security obligations require institutions to protect sensitive student performance data and comply with applicable data protection regulations. Additional limitations include the complexity of installation and ongoing maintenance, the inherent difficulty of evaluating code quality (as opposed to functional correctness) automatically, and the requirement for technical expertise to administer the judge microservice infrastructure.

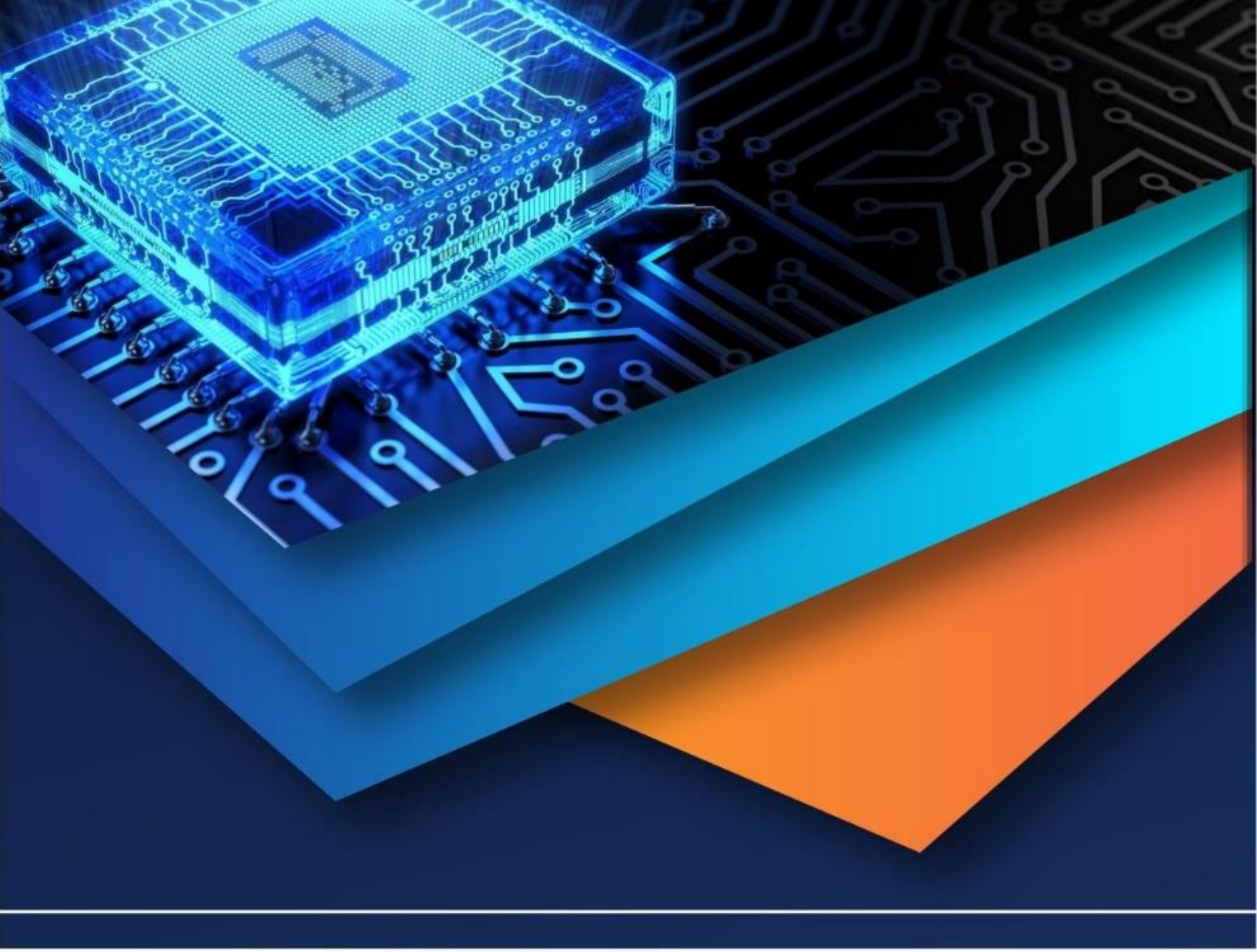
VI. VI. CONCLUSION AND FUTURE WORK

ExamNova successfully bridges the gap between LMS-based learning management and real-world coding assessment by embedding a Docker-sandboxed multi-language code judge directly within Moodle, coupled with a coding-specific analytics layer through IntelliBoard. The system demonstrates that secure, scalable, and pedagogically meaningful coding assessment is achievable entirely within an institutional LMS environment.

Future work includes plagiarism detection via MOSS or JPlag, an AI-powered hint system using LLM APIs, expanded language support for JavaScript, Rust, and Go, predictive analytics using machine learning on early coding metrics, and Kubernetes deployment for enterprise-scale scalability.

REFERENCES

[1] V. Daza, R. Traver, and C. Vidal-Castro, "Enhancing Moodle Insights: Leveraging Time Tracking Data Beyond Access Counts," IAIAI Letters on Learning Technology, vol. 3, no. 2, pp. 45–58, 2021.
[2] V. Daza and R. Traver, "Exploring Differences in Time Spent Tracking: IntelliBoard vs Edwiser Reports in Moodle," IAIAI Letters on Learning Technology, vol. 4, no. 1, pp. 12–24, 2022.
[3] A. Bakharia and S. Dawson, "Moodle for Learning Analytics and Institutional Research," IAIAI Journal of Educational Technology, vol. 18, pp. 88–102, 2020.
[4] A. Verbert, N. Manouselis, H. Drachsler, and E. Duval, "Learning Analytics in Moodle: A Systematic Review," Computers and Education, Elsevier, vol. 132, pp. 1–30, 2018.
[5] C.-H. Chang, G.-H. Tzeng, and D.-C. Lin, "Educational Data Mining and Learning Analytics in LMS," IEEE Transactions on Learning Technologies, vol. 10, no. 4, pp. 458–472, 2017.
[6] J. Grann and D. Bushway, "Dashboard-Based Learning Analytics in Higher Education," Proc. Springer Learning Analytics Workshop, Berlin, pp. 34–48, 2016.
[7] Y.-C. Tsai, H.-Y. Chen, and J.-C. Wang, "Container-Based Sandboxing for Online Code Judge Security," IEEE Access, vol. 7, pp. 105230–105242, 2019.
[8] M. Raja and S. Bhanu, "Integrating a Code Evaluation Module in Moodle for Programming Courses," International Journal of Engineering Education, vol. 36, no. 5, pp. 1678–1690, 2020.
[9] M. Meftah, H. Ziani, and Z. Bakkoury, "Improving Novice Programmer Experience with Browser-Based Code Editors," Proc. International Conference on Advanced Information Technology, Casablanca, pp. 113–118, 2021.
[10] D. Merkel, "Docker: Lightweight Linux Containers for Consistent Development and Deployment," Linux Journal, vol. 2014, no. 239, pp. 2–8, 2014.



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY