

LabPulse: An Intelligent Multi-Modal Framework for Virtual Laboratory, Automated Programming Assessment and Learning Analytics

Jiaxing Li¹; Ilapanda Pavan Kumar²; S. Mano Venkat³

^{1,2,3}School of Mechatronic Engineering, Xi'an Technological University, Xi'an, China

Publication Date: 2026/09/26

Abstract: Laboratory assessment in programming-oriented engineering courses combines executable artifacts, conceptual understanding, code quality, academic integrity, and submission discipline. Existing virtual laboratories and automated programming assessment tools commonly emphasize execution correctness and immediate feedback, while laboratory-assessment research identifies a continuing need to connect learning objectives with richer assessment evidence. This paper presents LabPulse, a prototype academic virtual laboratory and automated evaluation framework derived from an implemented React/Vite frontend and Express-based backend. The system organizes subjects and experiments, executes student programs through language-specific runner paths, evaluates submissions using a 100-mark multi-factor model, generates heuristic code-quality feedback, evaluates typed viva responses, screens code similarity, and exposes analytics/proctoring/help-request interfaces. The proposed Multi-Factor Laboratory Assessment Model (MFLAM) assigns 40 marks to test-case correctness, 25 to execution/code quality, 20 to viva-voce performance, and 15 to timeliness. A formal representation, assessment pseudocode, architecture, data-flow, research-gap analysis, and an empirical evaluation protocol are provided. Because no classroom dataset or measured user-study results are included in the supplied implementation, this paper reports implementation-derived capabilities rather than fabricated educational-effectiveness results. The resulting research agenda evaluates grading agreement, evaluation time, feedback latency, workload, student outcomes, and integrity-screening utility against conventional laboratory assessment.

Keywords: Virtual Laboratory, Automated Assessment, Programming Education, Code Evaluation, Viva-Voce, Learning Analytics, Code Similarity, Outcome-Based Education, Engineering Education.

How to Cite: Jiaxing Li; Ilapanda Pavan Kumar; S. Mano Venkat (2026). LabPulse: An Intelligent Multi-Modal Framework for Virtual Laboratory, Automated Programming Assessment and Learning Analytics. *International Journal of Innovative Science and Research Technology*, 11(9), 1529-1535. <https://doi.org/10.38124/ijisrt/26sep984>

I. INTRODUCTION

Engineering laboratory courses are expected to connect conceptual knowledge with practical performance. In programming laboratories, the practical artifact is executable source code, but competence also includes algorithmic reasoning, code organization, explanation of design decisions, and the ability to complete work within an academic schedule. Virtual laboratories can increase access and flexibility, and recent evidence reports positive effects on engineering learning outcomes, although virtual laboratories are better viewed as complements to hands-on experiences than universal replacements [10]–[12].

Automated programming assessment has a long research history. Reviews distinguish dynamic assessment, static analysis, testing, metrics, and feedback as complementary techniques [1]–[3]. More recent reviews show that automated graders often concentrate on correctness and unit testing, while fewer systems comprehensively assess maintainability, readability, documentation, or broader pedagogical

dimensions [8]. Automated feedback can provide rapid formative information, but its design affects how students use it and can create gaming behavior when feedback is too shallow [9].

LabPulse addresses this intersection by treating a laboratory submission as a multi-modal assessment object rather than a program-output pair. The implemented prototype combines experiment metadata, code execution, code-quality heuristics, typed viva evaluation, code-similarity screening, timeliness, and academic analytics. The contribution is therefore not a claim of a new compiler or a new plagiarism algorithm; it is an integrated assessment architecture and a configurable scoring model that combines multiple evidence streams into a laboratory grade.

The principal contributions are: (1) an implementation-grounded architecture for a virtual programming laboratory; (2) MFLAM, a weighted assessment formulation; (3) an assessment pipeline combining dynamic execution, heuristic static analysis, viva evaluation, and similarity screening; (4) a

research-gap analysis linking the prototype to prior work; and (5) an experimental protocol for validating reliability, efficiency, learning impact, and fairness without overstating unmeasured results.

II. RELATED WORK

Virtual laboratories. Reeves and Crippen's systematic review of undergraduate science and engineering virtual laboratories identified methodological limitations and a concentration on content-knowledge outcomes [11]. Nikolic et al. similarly found that laboratory research often emphasizes perceptions and cognitive measures, while objectives and assessments are not always clearly aligned [10]. Li and Liang's 2024 meta-analysis synthesized 46 studies from 22 publications and reported a positive overall effect for virtual laboratories in engineering education (Hedges' $g=0.686$), while also supporting their use as supplements rather than complete substitutes [12]. Frady's review further highlights accessibility and flexibility benefits alongside implementation and pedagogical challenges [18].

Automated programming assessment. Ala-Mutka surveyed static and dynamic approaches and emphasized speed, availability, consistency, and objectivity, while also noting the importance of pedagogical design [1]. Douce et al. reviewed test-based programming assessment and its evolution [2]. Ihantola et al. catalogued automatic assessment systems and identified diverse assessment strategies [3]. BOSS integrated correctness, quality metrics, plagiarism checking, submission management, and feedback [4], providing an important precedent for integrated assessment.

Insa et al. combined black-box and white-box assessment for Java exercises and reported substantial automation of assessment work in a real educational deployment [6]. Drop Project similarly integrates automated testing and quality assessment and reports extensive real-course use [7].

Feedback and AI. Keuning et al. reviewed 101 tools and found that automated feedback often identifies errors more readily than it explains how to fix them or what to do next [5]. Arakawa et al. experimentally studied feedback designs and found that the information conveyed by feedback materially affects learning interactions [9]. Messer et al.'s systematic review of 121 papers from 2017–2021 found that most tools assess correctness, with relatively limited assessment of maintainability, readability, and documentation [8]. Liu et al.'s 2025 systematic review places automatic evaluation, learning analytics, intelligent tutoring, and virtual experiment environments among major AI applications in engineering education [13].

Integrity and analytics. Source-code similarity has been studied through systems such as JPlag, which uses program-token comparison and Greedy String Tiling [16], and document fingerprinting methods such as Winnowing [17]. These works motivate treating similarity as evidence for human review rather than as an infallible plagiarism verdict. Learning analytics research frames educational data as a basis for understanding learning and supporting institutional decisions [14], [15]. LabPulse adopts this broader view by exposing OBE analytics, proctoring logs, and help-request interfaces in addition to submission scores.

III. RESEARCH GAP AND POSITIONING

Table I Research-Gap Positioning of LabPulse

Dimension	Prior Literature / Systems	LabPulse Position	Remaining Gap
Virtual lab	Access, simulation, flexibility; often separate from assessment [10]–[12]	Experiment-centered web workflow plus automated evaluation	Need empirical evidence of learning impact in programming labs
Program correctness	Dynamic tests and unit tests dominate [1]–[3], [8]	Public/hidden test-case evaluation	Need broader semantic and adversarial test coverage
Code quality	Static metrics appear but are less common [8]	Heuristic quality score and suggestions	Validate against expert rubric and reliable static-analysis tools
Conceptual assessment	Often separate from code grading	Typed viva integrated into total score	Validate keyword rubric against human viva scoring
Integrity	BOSS/JPlag and related methods support similarity analysis [4], [16], [17]	Token normalization + Jaccard risk screening	Calibrate thresholds and false-positive rates
Learning analytics	Well-established as a separate discipline [14], [15]	OBE analytics and telemetry endpoints	Map evidence to validated course outcomes
Integrated grading	Prior systems combine selected components [4], [7]	40/25/20/15 multi-factor laboratory grade	Measure reliability, fairness, and faculty workload

The table distinguishes implemented scope from unresolved empirical questions; it does not claim superiority over existing systems.

IV. SYSTEM ARCHITECTURE

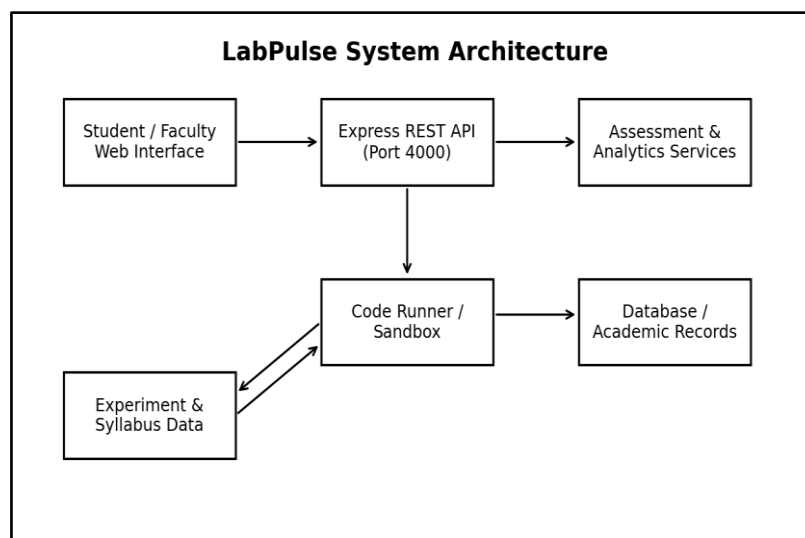


Fig 1 LabPulse Implementation-Oriented Architecture

The frontend is a React/Vite application styled with Tailwind CSS. The Vite configuration proxies /api requests to an Express backend on port 4000. The backend imports database, runner, grading, code-review, viva, plagiarism, experiment-generation, analytics, proctoring, and help-request services. The supplied source therefore follows a layered client/API/service/data organization rather than a monolithic browser-only design.

The implementation currently supports experiment and syllabus data for programming laboratories including Data Structures and Algorithms, Python Programming, Object-Oriented Programming (Java), Operating Systems & Systems, and Database Management Systems. Experiment metadata includes aims, theory, algorithms, input/output formats, starter code, test cases, and viva questions. For example, the first DSA experiment implements stack-based infix-to-postfix conversion and contains public and hidden test cases.

V. DATA FLOW

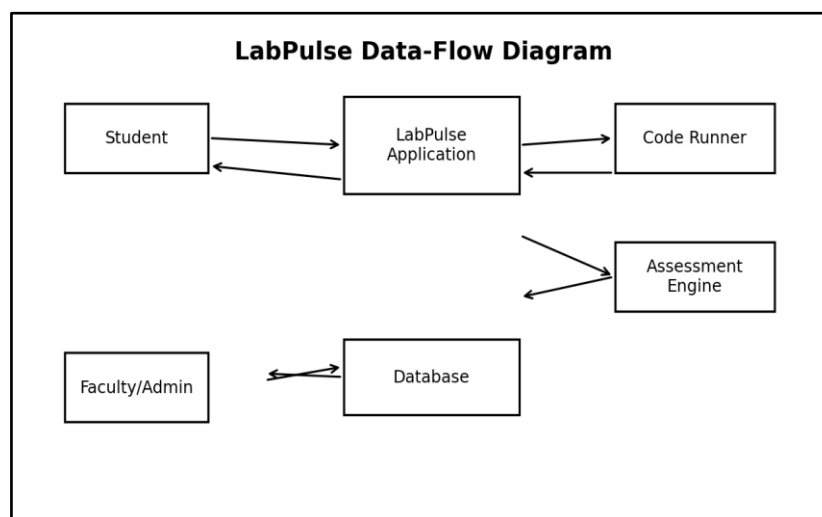


Fig 2 High-Level Data Flow for a Laboratory Submission

A typical student interaction begins with experiment selection and code entry. The submission is passed to the runner with language, source code, standard input, and timeout parameters. Execution results are then consumed by grading and code-review services. Viva responses are evaluated separately, similarity is computed across submissions for the same experiment, and the resulting evidence can be stored or surfaced through analytics and faculty-facing interfaces.

The implementation should be interpreted carefully: Python and JavaScript paths use process execution with timeout handling, while some C/C++ and Java paths are specialized or simulated for experiment families. Thus, the current prototype demonstrates an assessment architecture and selected execution pathways rather than a production-grade arbitrary-code sandbox.

VI. MULTI-FACTOR LABORATORY ASSESSMENT MODEL

LabPulse defines the final laboratory score as a weighted sum of four evidence dimensions:

$$S = C + Q + V + T, 0 \leq S \leq 100,$$

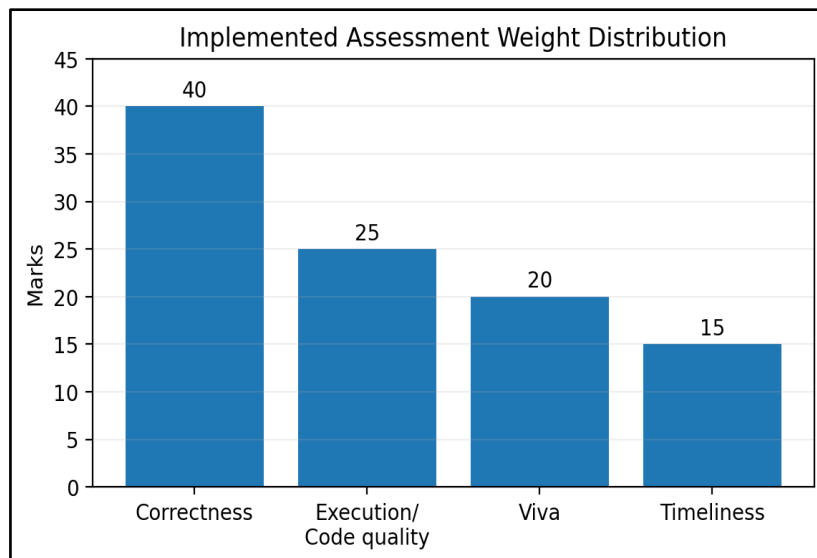


Fig 3 Implemented Assessment Weight Distribution; This is a Configuration, Not an Empirical Result

Correctness is proportional to passed test cases. The implementation records execution-related marks as a 25-mark component. Viva marks are derived from evaluated viva questions. Timeliness uses four states: EARLY, ON_TIME, GRACE_PERIOD, and LATE. Early and on-time submissions receive full timeliness marks; submissions within a 48-hour grace interval receive a decreasing score with an 8-mark floor; later submissions receive 5 marks.

This decomposition is intended to reduce over-dependence on a single observable. A program can be functionally correct but poorly explained; conversely, a student may demonstrate conceptual knowledge while submitting an incomplete artifact. The model makes those dimensions explicit and therefore enables separate empirical validation of each component.

VII. ASSESSMENT ALGORITHM

Algorithm 1 summarizes the implemented assessment logic.

Input: experiment E, submission code P, language L, test results R, viva answers A, deadline D, submission time t, peer submissions P*. Output: score S, grade G, evidence record.

- Execute P with the experiment-specific runner and collect pass/fail results, stdout, stderr, timeout state, and execution metadata.
- Compute C from the fraction of passed test cases, bounded to [0,40].
- Analyze code structure and execution evidence; compute Q∈[0,25].

where C∈[0,40] is test-case correctness, Q∈[0,25] is execution/code quality, V∈[0,20] is viva performance, and T∈[0,15] is timeliness. The implemented grade thresholds are O≥90, A+≥80, A≥70, B≥60, C≥50, and F<50.

- For each viva question, compare the typed response with expected conceptual keywords; aggregate to V∈[0,20].
- Compute $\Delta t = t - D$. If $\Delta t \leq 0$, set T=15. If $0 < \Delta t \leq 48$ h, reduce T according to the configured grace-period rule. Otherwise set T=5.
- Compute $S = \min(100, \max(0, C + Q + V + T))$ and map S to the configured letter-grade thresholds.
- For integrity screening, compare submissions belonging to the same experiment using normalized-token Jaccard similarity and assign LOW/MEDIUM/HIGH/CRITICAL risk bands.
- Return score breakdown, feedback, viva summary, timeliness state, similarity evidence, and timestamps.
- The pseudocode is deliberately separated from the integrity and analytics layers. A similarity alert should not automatically modify S without an institutional policy and human review process.

VIII. IMPLEMENTATION OF INTELLIGENT FEEDBACK

The supplied code-review component is best described as heuristic or rule-based intelligent analysis rather than a demonstrated large-language-model reviewer. It counts lines and loops, checks for recursion and common programming constructs, estimates time/space complexity using pattern rules, generates strengths and suggestions, and produces a code-quality score constrained to a 5–10 range. This distinction is important for scientific reproducibility and for avoiding an unsupported claim of generative-AI reasoning.

The viva module similarly uses structured questions and expected keywords. For each answer, matched and missing

keywords are recorded, a match percentage is calculated, and a discrete score is assigned using answer length and concept coverage. This is reproducible and inexpensive, but it is not equivalent to semantic evaluation by an expert examiner. The appropriate research question is therefore whether this rubric approximates expert judgments sufficiently for formative use or first-pass scoring.

IX. CODE SIMILARITY AND ACADEMIC INTEGRITY

The implementation normalizes source code by removing comments and whitespace, replacing strings and numeric literals with placeholders, tokenizing the result, and computing pairwise Jaccard similarity. Similarity thresholds are mapped to LOW (<50%), MEDIUM (50–69%), HIGH (70–84%), and CRITICAL (≥85%) risk. This is a screening mechanism, not a definitive plagiarism detector.

The design is consistent with the broader literature's use of structural similarity to surface suspicious pairs [4], [16], [17]. However, Jaccard similarity over token sets discards ordering and frequency information and can therefore produce both false positives and false negatives. A production evaluation should compare the current approach against a stronger baseline such as Greedy String Tiling or fingerprint-based methods and should report precision, recall, false-positive rate, and investigator agreement.

X. OUTCOME-BASED AND LEARNING ANALYTICS

The backend exposes an OBE analytics endpoint and additional telemetry endpoints for proctoring logs and help requests. These interfaces create the foundation for connecting individual assessment evidence with course-level outcome attainment. Learning analytics research emphasizes the value of turning educational interaction data into actionable evidence rather than merely accumulating logs [14], [15].

For research use, each experiment should be mapped to explicit course outcomes, and each assessment dimension should have a documented construct definition. A possible mapping is: correctness→application/problem solving; code quality→analysis/design; viva→conceptual explanation; timeliness→professional practice. Such mapping should be approved by the course team rather than assumed from the software implementation.

XI. EXPERIMENTAL DESIGN

The prototype currently provides no classroom dataset, controlled trial, or measured benchmark in the supplied files. Accordingly, no educational-effectiveness result is reported here. The following design is proposed to generate publishable evidence.

Table 2 Proposed Empirical Evaluation Design

Research Question	Independent Variable	Outcome Measure	Recommended Analysis
RQ1: Does LabPulse reduce evaluation time?	Manual vs LabPulse-assisted	Minutes/submission	Paired t-test or Wilcoxon; effect size
RQ2: Does automated scoring agree with experts?	Assessment mode	MAE, RMSE, ICC/correlation	Agreement and Bland–Altman analysis
RQ3: Does feedback arrive faster?	Manual vs automated	Feedback latency	Median/IQR; survival or time comparison
RQ4: Does feedback improve learning?	Intervention vs control	Pre/post score, resubmission success	ANCOVA / mixed model
RQ5: Is similarity screening useful?	Detector / threshold	Precision, recall, FPR	Confusion matrix + threshold analysis
RQ6: Is the model fair?	Student subgroup / context	Score distributions, calibration	Distributional and differential-performance analysis

Actual sample sizes, participant demographics, effect sizes, and p-values must be populated only after data collection.

XII. RESULTS REPORTING PLAN

A rigorous conference submission should report results at three levels. First, system-level measurements should include

API latency, runner timeout behavior, successful execution rate, and assessment throughput. Second, assessment validity should compare automated scores with blinded faculty scores on the same submissions. Third, educational outcomes should compare learning and feedback measures over a defined intervention period.

Table 3 Results Table to be Populated from an Empirical Study

Metric	Baseline	LabPulse	Effect / CI	Status
Median grading time per submission	TBD	TBD	TBD	To measure
Automated vs expert score MAE	TBD	TBD	TBD	To measure
Feedback latency	TBD	TBD	TBD	To measure
Similarity-screen precision	TBD	TBD	TBD	To measure
Student learning gain	TBD	TBD	TBD	To measure
Faculty workload	TBD	TBD	TBD	To measure

The use of TBD is intentional and prevents fabricated results.

XIII. LIMITATIONS AND THREATS TO VALIDITY

- Construct validity: heuristic complexity and code-quality rules are proxies, not complete static analysis or expert judgments.
- Internal validity: without randomized or matched groups, differences in outcomes may reflect instructor, cohort, or assignment effects.
- External validity: the current experiment catalog is programming-lab oriented; findings may not generalize to hardware, electronics, chemistry, or psychomotor laboratories.
- Measurement validity: typed-viva keyword matching can reward lexical overlap and miss semantically correct paraphrases.
- Integrity validity: Jaccard similarity is a screening heuristic and should not be used alone for disciplinary decisions.
- Security validity: process execution and simulated/specialized paths require hardening before untrusted multi-user production deployment; isolation, CPU/memory quotas, filesystem/network restrictions, secret management, and secure authentication are required.
- Reproducibility: the study must release anonymized benchmark assignments, test cases, scoring configuration, and evaluation scripts where permitted.

XIV. DISCUSSION

LabPulse is positioned as an integration framework for laboratory assessment rather than as a replacement for instructors. Its strongest research value is the explicit separation of evidence types: executable correctness, code quality, conceptual explanation, timing, and integrity evidence. This separation makes it possible to investigate which evidence dimensions correlate with learning outcomes and which should remain human-mediated.

The design also responds to a recurring concern in the literature: automated assessment can be technically effective while remaining pedagogically narrow [1], [5], [8], [10]. By incorporating viva, quality feedback, OBE analytics, and faculty intervention points, LabPulse creates a broader assessment surface. At the same time, the prototype should not be marketed as fully autonomous AI assessment until its heuristic modules are validated and any future generative models are explicitly documented.

XV. FUTURE WORK

- Replace or augment heuristic code analysis with AST-based static analysis and validated complexity estimation.
- Deploy isolated containers with strict resource, filesystem, and network controls for arbitrary student code.
- Introduce semantic viva evaluation with expert-validated rubrics and calibration studies.

- Compare token-set Jaccard screening with JPlag-style structural similarity and fingerprinting approaches.
- Develop a validated OBE mapping layer connecting experiments, rubrics, and course outcomes.
- Conduct a longitudinal controlled study covering learning gain, feedback behavior, grading agreement, and faculty workload.
- Add explainable analytics dashboards and audit trails for every automated score component.
- Evaluate fairness across programming languages, prior experience levels, and assignment types.

XVI. CONCLUSION

This paper presented LabPulse as an implementation-grounded framework for virtual programming laboratories and multi-modal automated assessment. The system combines experiment management, code execution, a 100-mark multi-factor assessment model, heuristic code review, typed viva evaluation, code-similarity screening, analytics, and faculty-support interfaces. The literature indicates strong interest in virtual laboratories, automated programming assessment, feedback, learning analytics, and AI-enabled engineering education, but also highlights gaps in pedagogical alignment, richer assessment coverage, and empirical validation [5], [8], [10]–[13]. LabPulse addresses these gaps at the architecture level, while the proposed experimental design defines the evidence needed to establish educational and operational effectiveness. The next research stage is therefore not to claim measured success, but to validate the framework through controlled, reproducible experiments against manual and existing automated assessment baselines.

REFERENCES

- [1]. K. M. Ala-Mutka, “A survey of automated assessment approaches for programming assignments,” *Computer Science Education*, vol. 15, no. 2, pp. 83–102, 2005, doi: 10.1080/08993400500150747.
- [2]. C. Douce, D. Livingstone, and J. Orwell, “Automatic test-based assessment of programming: A review,” *Journal on Educational Resources in Computing*, vol. 5, no. 3, Art. no. 4, 2005.
- [3]. P. Ihanola, T. Ahoniemi, V. Karavirta, and O. Seppälä, “Review of recent systems for automatic assessment of programming assignments,” in *Proc. 10th Koli Calling Int. Conf. Computing Education Research*, 2010, pp. 86–93, doi: 10.1145/1930464.1930480.
- [4]. M. Joy, N. Griffiths, and R. Boyatt, “The BOSS online submission and assessment system,” *Journal on Educational Resources in Computing*, vol. 5, no. 3, Art. no. 2, 2005, doi: 10.1145/1163405.1163407.
- [5]. H. Keuning, J. Jeuring, and B. Heeren, “A systematic literature review of automated feedback generation for programming exercises,” *ACM Transactions on Computing Education*, vol. 19, no. 1, Art. no. 3, 2018, doi: 10.1145/3231711.
- [6]. D. Insa, S. Pérez-Rubio, J. Silva, and S. Tamarit, “Semiautomatic generation and assessment of Java exercises in engineering education,” *Computer*

- Applications in Engineering Education, vol. 29, no. 5, pp. 1034–1050, 2021, doi: 10.1002/cae.22356.
- [7]. B. P. Cipriano, N. Fachada, and P. Alves, “Drop Project: An automatic assessment tool for programming assignments,” *SoftwareX*, vol. 18, Art. no. 101079, 2022, doi: 10.1016/j.softx.2022.101079.
- [8]. M. Messer, N. C. C. Brown, M. Kölling, and M. Shi, “Automated grading and feedback tools for programming education: A systematic review,” *ACM Transactions on Computing Education*, vol. 24, no. 1, Art. no. 10, 2024, doi: 10.1145/3636515.
- [9]. K. H. Arakawa, A. Turcan, T. Poehlman, and T. Greer, “Towards understanding the effective design of automated formative feedback for programming assignments,” *Computer Science Education*, vol. 32, no. 1, pp. 105–127, 2022, doi: 10.1080/08993408.2020.1860408.
- [10]. [10] S. Nikolic, M. Ros, K. Jovanovic, and Z. Stanisavljevic, “Remote, simulation or traditional engineering teaching laboratory: A systematic literature review of assessment implementations to measure student achievement or learning,” *European Journal of Engineering Education*, vol. 46, no. 6, pp. 1141–1162, 2021, doi: 10.1080/03043797.2021.1990864.
- [11]. S. M. Reeves and K. J. Crippen, “Virtual laboratories in undergraduate science and engineering courses: A systematic review, 2009–2019,” *Journal of Science Education and Technology*, vol. 30, pp. 16–30, 2021.
- [12]. J. Li and W. Liang, “Effectiveness of virtual laboratory in engineering education: A meta-analysis,” *PLoS ONE*, vol. 19, no. 12, e0316269, 2024, doi: 10.1371/journal.pone.0316269.
- [13]. C. Liu, G.-C. Wang, and H.-F. Wang, “The application of artificial intelligence in engineering education: A systematic review,” *IEEE Access*, vol. 13, pp. 17895–17910, 2025, doi: 10.1109/ACCESS.2025.3532595.
- [14]. C. Romero and S. Ventura, “Educational data mining and learning analytics: An updated survey,” *WIREs Data Mining and Knowledge Discovery*, vol. 10, no. 3, e1355, 2020, doi: 10.1002/widm.1355.
- [15]. G. Siemens, “Learning analytics: The emergence of a discipline,” *American Behavioral Scientist*, vol. 57, no. 10, 2013, doi: 10.1177/0002764213498851.
- [16]. L. Prechelt, G. Malpohl, and M. Philippsen, “Finding plagiarisms among a set of programs with JPlag,” *Journal of Universal Computer Science*, vol. 8, no. 11, pp. 1016–1038, 2002, doi: 10.3217/jucs-008-11-1016.
- [17]. S. Schleimer, D. S. Wilkerson, and A. Aiken, “Winnowing: Local algorithms for document fingerprinting,” in *Proc. ACM SIGMOD Int. Conf. Management of Data*, 2003, pp. 76–85, doi: 10.1145/872757.872770.
- [18]. K. Frady, “Use of virtual labs to support demand-oriented engineering pedagogy in engineering technology and vocational education training programmes: A systematic review of the literature,” *European Journal of Engineering Education*, vol. 48, no. 3, 2023, doi: 10.1080/03043797.2022.2141610.
- [19]. B. S. Bloom, Ed., *Taxonomy of Educational Objectives: The Classification of Educational Goals*. New York, NY, USA: Longmans, Green, 1956.