Journal of Systemics, Cybernetics and Informatics (2025) 23(5), 5-10
https://doi.org/10.54808/JSCI.23.05.5

AI-Driven Grading and Moderation for Collaborative Projects in Computer Science Education

Songmei YU
Department of Computer Science
School of Business and Information Sciences
Felician University, Rutherford, NJ 07070

Andrew ZAGULA
Bridgewater-Raritan High School
600 Garretson Rd, Bridgewater, NJ 08807

ABSTRACT

Collaborative group projects are integral to computer science education, fostering teamwork, problem-solving, and industry-relevant skills. However, assessing individual contributions within group settings is a long-standing challenge. Traditional assessment strategies, such as equal distribution of grades or subjective peer assessments, fall short in terms of fairness, objectivity, and scalability, especially in large classrooms. This paper introduces a semi-automated, AI-assisted grading system that evaluates both project quality and individual effort using repository mining, communication analytics, and machine learning models. The system comprises modules for project evaluation, contribution analysis, and grade computation, integrating seamlessly with platforms like GitHub. A pilot deployment in a senior-level course demonstrated high alignment with instructor assessments, increased student satisfaction, and reduced instructor grading effort. We conclude by discussing implementation considerations, ethical implications, and proposed enhancements to broaden applicability.

Keywords: Collaborative Group Projects, Assessment, AI-Assisted Grading System

1. INTRODUCTION

Group projects are foundational to computer science curricula, offering students hands-on experience in software engineering practices such as Agile methodologies, version control, collaborative coding, and iterative testing. These collaborative endeavors mirror professional environments, making them invaluable in preparing students for real-world development workflows. However, assigning fair individual grades remains difficult, particularly when group sizes are large or contributions are unevenly distributed.

Instructors often lack granular visibility into team dynamics, leading to assessments based on incomplete information. Peer evaluations, though valuable, can be influenced by bias, social dynamics, and non-academic considerations. Manual oversight becomes unfeasible at scale, prompting the need for automated tools that can provide accurate, transparent, and scalable assessments.

The core objectives of this research are to design, implement, and evaluate an AI-based grading and moderation system that (1) evaluates overall group project quality using standardized and customizable metrics, (2) quantifies individual contributions using objective data sources like version control and communication logs, (3) generates fair and explainable individual grades, (4) reduces instructor workload while maintaining oversight and flexibility, and (5) enhances transparency and fairness, thereby improving student satisfaction.

2. RELATED WORK

Traditional grading methods include equal grading (all group members receive the same grade regardless of contribution), peer evaluation tools, and instructor observation. Peer evaluation systems like CATME and SPART+ help assess team member performance from social and task-based perspectives. However, these systems are susceptible to bias, collusion, and peer pressure [1][9]. Additionally, peer reviews may fail to reflect real technical contributions, especially in large or distributed teams [10].

Instructor observation attempts to gauge individual effort based on limited interactions, project meetings, or code reviews. Yet, this method is time-consuming, not scalable, and lacks consistency, particularly in large classes or remote settings [2][11].

To overcome these limitations, recent research has explored mining software repositories to estimate contribution. Metrics such as number of commits, lines changed, file ownership, and authorship of specific code blocks offer quantifiable proxies for student effort [3][4][12][13]. However, committing frequency or volume alone does not always correlate with meaningful contributions, as students may make frequent but trivial changes (e.g., whitespace edits or repeated formatting). Some tools, such as GitGrade, Codequiry, and Gitorium, provide basic metrics from version control logs but lack integrated evaluation frameworks that account for quality, context, or collaboration patterns. More advanced solutions apply Natural Language Processing (NLP) to mine issue trackers, pull request discussions, and documentation comments [5][14][15]. This yields a more holistic view of individual contributions, capturing leadership roles, communication, and task complexity—not just code volume.

The broader field of Artificial Intelligence in Education (AIEd) applies machine learning, educational data mining, and analytics to support adaptive instruction, feedback, and engagement monitoring [6]. However, automated grading systems face

criticism due to lack of transparency, limited adaptability, and reliance on historical patterns, which may disadvantage atypical students.

Finally, ethical considerations such as algorithmic fairness, data privacy, surveillance, and student consent are critical [8]. These must be carefully addressed before integrating AI into high-stakes assessments like course grades.

3. SYSTEM DESIGN AND ARCHITECTURE

In this paper, we propose an AI grading system, which consists of three core modules: Project Quality Assessment Module (PQAM), Individual Contribution Analyzer (ICA), and Grading Engine (GE). We will illustrate each one and how they interact with each other with details in the following sections.

3.1 Project Quality Assessment Module (PQAM) This module is a comprehensive evaluation component designed to assess both the technical and qualitative integrity of a group software project. It leverages a range of automated tools and AI models to ensure the project meets academic and industry standards across multiple dimensions. The module supports objective grading, encourages best practices, and reduces manual workload for instructors. This module has five submodules which include Code Quality, Testing Coverage, Documentation, Functionality and Usability. The final output from this module is a general Project Quality Score (PQS), representing the overall quality of the group's project. Each module by default has the same weight. We can make weights configurable per project based on learning outcomes and project context.

Code Quality submodule uses static analysis to examine the source code without executing it. The analysis helps identify structural flaws, maintainability issues, and stylistic violations. It first builds Complexity Metrics that includes (1) Cyclomatic Complexity, it measures the number of independent paths through the code. A high value suggests complex, harder-to-maintain code, and (2) Halstead Metrics, it quantifies computational complexity based on operators and operands. These values offer insight into code readability, maintainability, and error-proneness.

Second, it uses Style Adherence to check the code compliance with certain programming language standards. For example, we can use Python tools like flake8, pylint, and black are used to check compliance with PEP8 standards. For JavaScript, tools like ESLint or Prettier ensure the code follows established best practices and organizational rules.

Third, it uses Code Duplication Detection to apply certain tools such as SonarQube, PMD, or jscpd and detect copy-pasted blocks of code, which indicate poor abstraction and reusability practices.

Testing Coverage submodule is crucial to project reliability as a robust testing practice evaluates how extensively the codebase is covered by tests. The available coverage tools for Python could be pytest-cov or coverage.py that can generate reports online, branch, and path coverage. For Java, JaCoCo or JUnit integration helps assess method and class-level coverage. For JavaScript, Istanbul/nyc or Jest produce detailed test coverage maps. Furthermore, beyond just counting lines covered, some AI models or advanced scripts can analyze whether edge cases, exception handling, and input validations are adequately tested to enhance the depth of coverage.

Documentation submodule reflects a team's ability to communicate and maintain their software. This part uses NLP (Natural Language Processing) techniques to assess the quality and relevance of project documentation. For example, for readability and clarity, we can use NLP models like BERT, spaCy, or GPT-based classifiers to assess grammar, structure, and coherence of README files and manuals. For completeness check, we can use heuristics and AI models that verify the inclusion of essential elements such as installation steps, usage instructions, architecture overview, contributor guidelines, and licensing. For inline comments & docstrings, we can use parser tools like Docformatter, Doxygen, or Sphinx to assess whether functions/classes are documented or perform the semantic similarity analysis that checks whether the comment matches the code logic or is superficial. For project Wikis or notebooks, Markdown or Jupyter, Notebook-based documentation is evaluated for technical depth, clarity of thought, and instructional value.

Functionality submodule is a direct evaluation of whether the software works as intended. We can enable an Automated Test Execution process, where predefined or student-supplied unit tests and integration tests are executed in sandboxed environments (e.g., Docker containers). The grading system compares actual outputs with expected outcomes and flags deviations. If students use tools like GitHub Actions, Travis CI, or GitLab CI, the system checks for passing build statuses and continuous testing practices. Moreover, feature validation can be conducted by using AI-based log analysis or semantic parsing of the project requirements to match implemented functionality against deliverables.

Usability submodule assesses how intuitive, responsive, and accessible the system is, particularly if the project includes a GUI or web-based interface. There are several major components that can evaluate code usability. For example, heuristic evaluation (optional/manual augmentation) can apply usability heuristics (e.g., Nielsen's 10 principles) to detect issues like inconsistent navigation, lack of feedback, or poor error messaging. UI testing frameworks such as Selenium, Cypress, or Puppeteer can be used to run scripted tests across various devices and browsers. For accessibility checks, Lighthouse, axe-core, or Pa11y can be integrated to check for WCAG (Web Content Accessibility Guidelines) compliance. Color contrast, screen reader compatibility, and keyboard navigation are some of the key elements analyzed. For responsiveness analysis, UI components are tested across different screen resolutions to verify mobile and desktop compatibility.

In general, Project Quality Assessment Module (PQAM) (refer to Figure 3-1) serves as a holistic evaluation engine that combines static code analysis, test validation, NLP-based documentation review, and UI/UX assessments. It fosters best engineering practices and enables fair, scalable, and transparent grading of collaborative software projects in computer science education.

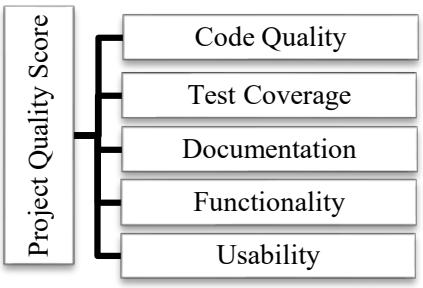

Figure 3-1: Project Quality Assessment Module

3.2 Individual Contribution Analyzer (ICA)

This analyzer is a core module designed to objectively assess each student's involvement in a collaborative group software project. Traditional group assessments often obscure individual effort, leading to potential unfairness in grading. ICA tackles this by analyzing multi-faceted signals from the development lifecycle to quantify and qualify each student's contribution. The ICA operates across four key dimensions: Commit History Analysis, Code Ownership, Issue Tracker Participation, and Code Review Evaluation.

Commit History Analysis evaluates version control activity, with a focus on *meaningful code contributions* rather than superficial edits. First, it filters trivial changes. AI heuristics and regex rules filter out non-substantive changes such as whitespace adjustments, minor comment updates, or renaming variables. This prevents students from artificially inflating their contribution through "commit spam." Second, for temporal and behavioral analysis, the system can observe patterns such as frequency of commits, time of day, and burst activity (indicating rushed contributions). Clustering algorithms (e.g., DBSCAN) may be used to detect organic vs. last-minute participation. Third, quantitative metrics such as line additions, deletions, and net code changes are calculated. Furthermore, special weighting can be applied to the commits involving test cases, documentation, or refactors.

Code Ownership attributes authored code lines to individuals by leveraging the git blame command. In more detail, every line of code in the final submission is mapped to its author using Git history. Edited lines are tracked across commit history to identify both original authors and significant editors. Full ownership is assigned to those who write or significantly revise code logic. Partial credit is given for minor edits or formatting adjustments. Code ownership is aggregated at the function/class/module level to identify who owned critical parts of the system (e.g., database model vs. frontend UI).

Issue Tracker Participation captures non-code contributions, which are often overlooked in traditional grading. First, the system tracks who created issues (e.g., bug reports, feature requests) and who commented, resolved, or referenced them in commits. This can work with GitHub Issues, GitLab, Jira, or any integrated issue tracking system. Also, NLP tools parse issue labels to identify contribution type: "documentation," "performance," "bug fix," "enhancement," etc. Also, the system can collect engagement metrics, for example, count and categorize interactions (e.g., number of issues raised/resolved), and measures responsiveness to team comments or review requests.

Code Review Evaluation is a key component in peer code review, which is critical for software quality and learning, and ICA evaluates how constructively and frequently each student participates. The system can conduct the following analysis, if not all, to allow peer code review. The first one is Review Frequency, where the number of pull request reviews and comments per student is tracked, and the active participation in feedback loops (suggesting improvements, identifying flaws) is noted. The second one is NLP-based Depth Analysis that uses NLP models (e.g., BERT, RoBERTa) to evaluate whether a review identifies logic issues, or suggests improvements, or refers to standards or best practices. Here the surface-level or generic comments receive lower weighting. The third one is Sentiment and Tone Analysis to detect whether reviews are constructive, neutral, or overly critical/toxic. This encourages professionalism and positive collaboration.

Finally, the scoring and visualization of ICA adopts multimodal aggregation. ICA uses weighted rubric to combine scores from the four components into a contribution index for each student. Customizable weights allow instructors to emphasize code vs. communication vs. design. A visualization dashboard with contribution heatmaps, pie charts, and timelines help visualize who contributed to which files, the evolution of each student's work overtime and participation spikes (e.g., just before deadlines). Furthermore, anomaly detection can be integrated into ICA where AI can flag potential freeloading or over-contributors, and team imbalance alerts can prompt early intervention by instructors.

Overall, the Individual Contribution Analyzer (ICA) (refer to Figure 3-2) offers a transparent and data-driven way to assess individual efforts in group projects. By analyzing both technical and collaborative behaviors, ICA promotes fairness, accountability, and reflective learning in collaborative computer science education.

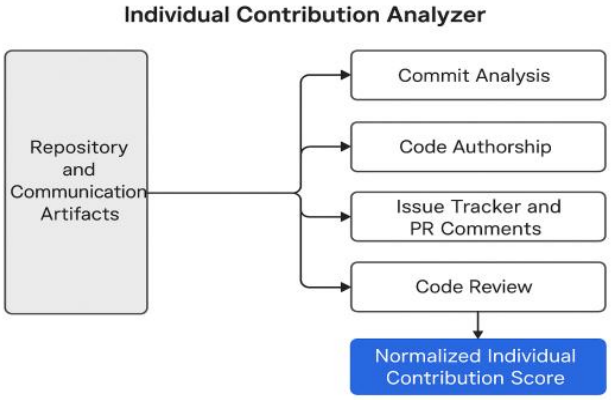

Figure 3-2: Individual Contribution Analyzer

3.3 Grading Engine (GE)

The Grading Engine (GE) is the final layer of the AI-driven assessment pipeline. It performs weighted aggregation, normalization, anomaly detection, and score interpretation to generate final individual grades from collective group output.

The GE calculates each student's final score by combining Project Quality Score (PQS) output from PQAM, representing the overall technical quality of the group's work, and Normalized Contribution Score (NCS), output from ICA, representing each student's individual input relative to their peers.

The default formula is as follows:

Final Grade = (PQS × 0.6) + (NCS × 0.4) **Eq. (1)**

This formula is configurable. Instructors can adjust the weightings to emphasize team output (e.g., 80/20) or individual effort (e.g., 50/50), depending on pedagogical goals. Grades are automatically generated but flagged for manual review if anomalies (e.g., outliers in contribution) are detected.

To ensure fair grading across teams of varying sizes and dynamics, ICA outputs could be normalized. For example, we can use Z-score Normalization or Min-Max Scaling, where each student's raw contribution score is scaled within the context of their team to allow comparison and mitigate impact from team composition. Also, extremely low contributions (e.g., <10%) may trigger floor limits, and extremely high contributions may be capped to prevent "grade hogging" and support equity. Instructors can optionally add bonuses for exemplary contributors or penalties for under-participation based on team evaluations or reflections.

The GE automatically detects irregularities or unfair patterns and flags them for human intervention. For example, it uses statistical models (e.g., interquartile range, standard deviation) to detect contributors with far higher/lower input than peers, or teams with extreme PQAM scores but low ICA engagement. Inconsistencies can also be caught such as mismatches between contribution levels and code authorship (e.g., frequent commits with minimal actual authorship), or imbalanced participation in code reviews or issues. Flagged submissions are queued in an instructor dashboard. The instructor is presented with visualizations of contribution breakdowns, suggested rubric-based adjustments and space for human notes or overrides.

A key goal of GE is to promote clarity and fairness in grading. It will provide breakdown reports for students that show the PQAM score, NCS, and final grade, with possible visuals including bar graphs, radar charts, and contribution timelines. It also provides feedback prompts that can auto-generate personalized feedback summaries, for example, "Your code coverage and documentation were strong, but ode review participation was low.", or "Project quality was excellent, but individual contribution was significantly below team average." All grading decisions and overrides are logged for transparency in case of appeals or disputes.

The Grading Engine (GE) acts as the bridge between AI-driven analysis and practical grading outcomes. By combining group quality and individual contribution metrics with anomaly detection and transparency features, GE ensures that every student receives a fair, evidence-based grade—while still allowing room for instructor judgment when necessary.

4. SYSTEM IMPLEMENTATION AND ANALYSIS

To validate the effectiveness, fairness, and scalability of the AI-based grading system, a pilot deployment was conducted during the Fall 2024 Software Engineering course at a small-sized university. The course comprised 20 students divided into 5 teams, working on collaborative projects over an 8-week schedule. Projects adhered to Agile methodologies and required the use of GitHub for version control, issue tracking, and code submissions. This system was implemented in Python, utilizing frameworks such as Flask for RESTful API development and PostgreSQL for relational data persistence. The frontend dashboards were created using React.js, allowing instructors and students to interact with project metrics, grades, and feedback visually. The key libraries and tools include (1) Scikit-learn, TensorFlow – for building machine learning models (e.g., contribution prediction, NLP classification), (2) Radon, Pylint – static code analysis for metrics such as cyclomatic complexity and style compliance, (3) GitPython – used to interface with Git repositories for extracting commit history and file authorship data, and (4) SpaCy, NLTK – employed in the NLP pipeline to process README files, code comments, and issue tracker text for evaluating documentation and collaboration.

4.1 Workflow Overview

The system followed a modular 5-step pipeline.

Step 1: Project Submission
Students pushed final codebases and documentation to GitHub Classroom, ensuring version control and consistent submission structure.

Step 2: Data Extraction
Repositories were programmatically mined using the GitHub REST API and GraphQL endpoints, extracting commit history and diffs, and issue tracker logs, Pull requests and code reviews. As well as README files, wikis, and inline comments.

Step 3: Model Evaluation
PQAM evaluated group-level quality using static analysis, test coverage metrics, documentation assessment, and usability checks. ICA parsed Git data and communication records to infer individual contributions using attribution modeling, NLP, and statistical analysis.

Step 4: Grade Calculation
The Grading Engine (GE) combined PQAM and ICA outputs via a configurable formula. Anomaly detection algorithms flagged potential freeloaders or disproportionate contributions for manual review.

Step 5: Dashboard Review
Instructors accessed a secure web portal to view breakdowns of team performance and individual contribution, adjust or approve final grades, and export results to the LMS.

A system flowchart was generated (refer to Figure 4.1), illustrating modular interactions from submission through evaluation, grading, and review. The diagram helps stakeholders understand the end-to-end automation and where human judgment is involved.

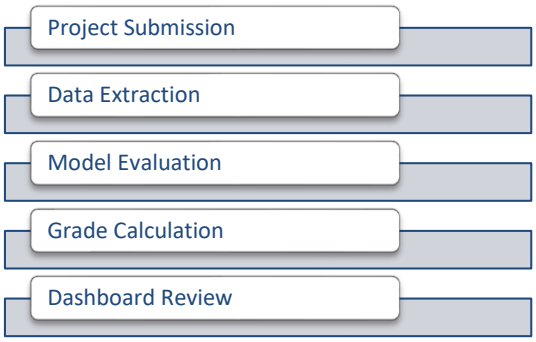

Figure 4.1: AI Grading System Workflow

4.2 Metrics and Evaluation Methods

To assess system validity and impact, three key methods were employed.

The first is Instructor Alignment. AI-generated grades were compared against traditional instructor-assigned grades using Pearson correlation (r) to assess consistency.

The second one is student perception. A post-project survey (N=53) evaluated students' views on fairness, transparency, and satisfaction using a 5-point Likert scale.

The last one is instructor effort. Time spent on grading was logged and compared with previous semesters using manual rubrics.

4.3 Pilot Results and Analysis

The AI system showed a strong alignment with human grading decisions: Pearson r = 0.91, suggesting high validity. Student Satisfaction Survey shows Fairness: 4.3/5, Transparency: 4.5/5. Also, it showed ~45% reduction in grading effort compared to manual evaluation methods. Finally, 12% of cases required manual intervention (e.g., for flagged anomalies, non-code contributions).

Three anonymized cases illustrate the system's effectiveness in capturing real-world dynamics. First, contributions were evenly distributed (within ±5%), which means AI-assigned grades were uniform and required no manual override. Second, one team member authored 58% of the production code and conducted most issue resolution. The system elevated their individual score accordingly, with instructor validation. Third, one student inflated activity with frequent whitespace and comment-only commits. The ICA module, using code diff analysis, down weighted these and flagged the case, avoiding unfair grade inflation.

This pilot result shows three advantages of using AI grading system in the collaborate coding projects. First, it shows the scalability. It seamlessly managed a class of 20 students and could scale to 200+ with minimal performance degradation. It also shows objectivity as it replaced subjective perceptions with data-driven, consistent evaluation. Finaly, it shows transparency as all students (based on an after-class survey) appreciated visibility into how grades were computed, improving trust in assessment. At the same time, it improves the engagement incentive because awareness of tracked metrics motivates students to participate consistently.

However, we are facing certain challenges if we fully rely on the system to grade. Important activities such as design ideation, presentation preps, and team facilitation were harder to quantify objectively. Also, students who failed to consistently use GitHub or issue trackers had underrepresented contributions. Moreover, students with weaker English skills or non-native grammar patterns risked lower documentation scores. Ongoing calibration is needed to avoid bias.

There are certain ethical considerations too**.** Students should be informed of the evaluation process and shown the grading rubrics and algorithmic logic before the class starts. From a data privacy perspective, the system should follow FERPA-compliant practices, ensuring data access was restricted to instructors and the academic institution. Also, students could contest grades via a structured appeal process. Manual override tools ensured fairness was not compromised by automation.

4.4 Future Directions

To extend functionality and improve adaptability, several research and engineering enhancements are planned. Each enhancement is an individual research topic or implementation effort. There is a lot of work ahead, we highlight several major directions that need to be addressed in the future.

IDE Integration: develop plugins for popular Integrated Development Environments (IDEs) such as Visual Studio Code, IntelliJ, and Eclipse to track local coding behavior, file ownership, and collaboration patterns.

This feature would enable seamless data collection from developers' working environments, offering insight into how and when students contribute to a shared project. By tracking keystrokes, file changes, commit frequencies, and time spent per file, these plugins can attribute contributions more accurately and detect patterns such as code ownership, multitasking, and collaboration bursts. It also enhances the Individual Contribution Analyzer (ICA) by providing rich metadata beyond Git history. Major challenges include preserving student privacy, managing data volume, and ensuring cross-platform compatibility.

Multimedia Artifact Analysis: use machine learning models to evaluate non-code artifacts, including screenshots, wireframes, presentations, and videos.

Many software engineering and CS capstone projects include deliverables like UI mockups, slide decks, and demo videos. Traditional grading overlooks these components or evaluates them manually. By applying computer vision, natural language processing, and audio analysis techniques, the system could automatically assess the clarity, completeness, and professionalism of these artifacts. For example, image recognition could assess consistency in UI wireframes; speech analysis could evaluate clarity and fluency in demo videos. This expansion acknowledges the holistic nature of project work and emphasizes design and communication skills alongside code.

Enhanced Peer Review: combine subjective peer feedback with objective ICA metrics to produce more balanced and trustworthy individual assessments.

Peer reviews offer a unique perspective into team dynamics and can capture nuances that automated systems miss, such as leadership, initiative, or teamwork quality. However, they can be biased or inconsistent. By integrating peer evaluations with ICA's behavioral metrics (e.g., code contributions, task tracking, communication patterns), the platform can cross-validate data, flag anomalies, and calibrate scores. Machine learning models may even weigh peer input differently based on historical reviewer reliability, fostering a more trustworthy and transparent system.

Adaptive Grading Models: apply reinforcement learning or similar techniques to dynamically adjust grading models based on past performance and instructor feedback.

Current grading systems rely on fixed weights (e.g., 60% project quality, 40% individual contribution). Adaptive grading would instead learn from instructor corrections—such as grade adjustments or override patterns—and iteratively improve the scoring logic. Over time, the system can learn how to handle edge cases, identify scoring inconsistencies, and personalize grading for different course contexts. This reduces manual grading effort and improves grading accuracy, especially in large or diverse classes.

Cross-Disciplinary Expansion: adapt the platform for use in disciplines beyond computer science, such as business, design, humanities, and engineering.

Many academic fields assign collaborative group projects but lack structured tools to assess them effectively. By abstracting core features (e.g., contribution tracking, artifact analysis, peer review), the platform can support project-based learning in various domains. For example, in design, it may evaluate the evolution of creative assets; in business, it could track contributions to a business plan or pitch deck. This would require customizing assessment models, interfaces, and rubrics to align with discipline-specific expectations and workflows.

Bias Audits: implement regular fairness assessments to detect and mitigate grading bias across gender, language, or demographic groups.

AI-powered assessment systems risk encoding or amplifying biases—especially when evaluating subjective artifacts or using peer input. Conducting systematic audits using fairness metrics (e.g., disparate impact, equal opportunity) helps detect whether certain groups are under- or over-represented in top scores. This could involve anonymizing submissions during evaluation, monitoring statistical parity, and incorporating fairness-aware ML models. Transparent reporting and instructor tools to review flagged cases would promote ethical and equitable grading.

LMS & Rubric Integration: enable direct export of grades, comments, and rubrics into Learning Management Systems (LMS) like Canvas, Moodle, Brightspace, and Blackboard.

For broad classroom adoption, the system must integrate with existing instructor workflows. By aligning with LMS APIs, the platform can automate the push of final grades, individualized feedback, and rubric-based assessments, save time and reduce data entry errors. Furthermore, bidirectional integration may allow the system to pull course context, deadlines, and participation data to inform grading. This streamlining is essential for scaling use across institutions and ensuring consistency with institutional grading policies.

5. CONCLUSION

This paper presents a robust, AI-powered grading system that addresses the key challenges in assessing collaborative group projects in computer science curriculum. By leveraging data from code repositories, communication logs, and automated analysis tools, the system ensures greater fairness, scalability, and insight. Initial trials confirm its efficacy and positive reception. Future expansions will aim to handle non-code contributions, multimedia artifacts, and adaptive learning across disciplines. With ethical safeguards in place, AI-driven grading holds transformative potential for project-based learning in computer science and beyond.

REFERENCES

[1] M. W. Ohland et al., "The comprehensive assessment of team member effectiveness (CATME)," Acad. Manage. Learn. Educ., vol. 11, no. 4, pp. 609–630, 2012.

[2] L. Freeman and L. Greenacre, "An examination of socially destructive behaviors in group work," J. Mark. Educ., vol. 33, no. 1, pp. 5–17, 2011.

[3] C. Bird, D. Pattison, R. D'Souza, V. Filkov, and P. Devanbu, "Determining code ownership," in Proc. 31st Int. Conf. Softw. Eng. (ICSE), 2009, pp. 603–613.

[4] D. Spinellis and V. Giannikas, "A platform for collaborative software engineering education," IEEE Trans. Educ., vol. 55, no. 4, pp. 445–452, 2012.

[5] J. Tsay, L. Dabbish, and J. Herbsleb, "Let's talk about it: Evaluating contributions through discussion in GitHub," in Proc. 22nd ACM SIGSOFT Int. Symp. Found. Softw. Eng. (FSE), 2014, pp. 144–154.

[6] G. Siemens, "Learning analytics: The emergence of a discipline," Amer. Behav. Sci., vol. 57, no. 10, pp. 1380–1400, 2013.

[7] S. Singh and J. Holt, "Applying NLP to evaluate software documentation," in Proc. 41st Int. Conf. Softw. Eng. Companion (ICSE-Companion), 2019, pp. 374–375.

[8] B. Williamson, Big Data in Education: The Digital Future of Learning, Policy and Practice. London, U.K.: SAGE, 2017.

[9] D. B. Kaufman, R. M. Felder, and H. Fuller, "Peer ratings in cooperative learning teams," in Proc. ASEE Annu. Conf. Expo., 2000.

[10] J. Hamer et al., "Tools for contributing student programming effort," in Proc. 10th Annu. SIGCSE Conf. Innov. Technol. Comput. Sci. Educ., 2005, pp. 55–59.

[11] E. F. Gehringer, "Strategies and mechanisms for electronic peer review," in Proc. 31st Front. Educ. Conf. (FIE), 2001, vol. 3, pp. S2C–9.

[12] B. Vasilescu et al., "How do developers blog?" in Proc. 10th Joint Meeting Found. Softw. Eng., 2015, pp. 449–460.

[13] P. C. Rigby and M.-A. Storey, "Understanding broadcast-based peer review on open-source software projects," in Proc. 33rd Int. Conf. Softw. Eng. (ICSE), 2011, pp. 541–550.

[14] S. Wang, D. Lo, and L. Jiang, "An empirical study on developer interactions in GitHub," in Proc. 23rd Int. Conf. Softw. Anal., Evol. Reeng. (SANER), 2016, pp. 1–12.

[15] D. W. McDonald and M. S. Ackerman, "Just talk to me: A field study of expertise location," in Proc. CSCW, 1998, pp. 315–324.