Automated Rubric-Based Classification of Student Peer Code Review Feedback

Theresia Devi Indriasari* and Yohanes Sigit Purnomo W.P.

Department of Informatics, Faculty of Industrial Technology, Universitas Atma Jaya Yogyakarta, Yogyakarta, Indonesia
Email: devi.indriasari@uajy.ac.id (T.D.I.); sigit.purnomo@uajy.ac.id (Y.S.P.W.P.)
*Corresponding author

Manuscript received September 16, 2025; revised November 20, 2025; accepted December 26, 2025; published May 15, 2026

***Abstract*—Student peer code review feedback often does not align with rubric criteria. This reduces its value for learning. This study develops automated methods to classify student peer code review feedback in Bahasa Indonesia. The classification uses seven labels. Six labels represent rubric criteria for code quality in introductory programming courses, namely Variable for naming clarity, Expression for expressions and data types, Control Flow for program logic and exception handling, Comments for code documentation, Layout and Formatting for readability and structure, and Decomposition for modularization and task separation. One additional label captures general feedback. A dataset of 2281 student feedback was created. The data were collected through peer code review activities in an introductory programming course at a higher education institution. The dataset was validated with high agreement between raters (Cohen's Kappa = 0.9463). Three methods were tested: machine learning, deep learning, and few-shot prompting with large language models. Random Forest with count vectorization gave the best results. It reached an F1-score of 0.9430. This result was higher than that of recurrent convolutional neural networks with FastText embeddings (F1-score = 0.9113) and few-shot prompting (F1-score = 0.852). The results show that classical machine learning with token features can be more effective than complex models. These findings provide a foundation for integrating automated classification into peer code review tools, supporting more consistent, rubric-based feedback in computing education.**

***Keywords*—student peer code review, rubric-based classification, machine learning, deep learning, few-shot prompting, computing education, Bahasa Indonesia**

I. Introduction

Peer review is a collaborative learning approach that helps students give and receive feedback. It supports active learning, strengthens critical thinking, and improves communication skills [1–5]. Through this process, students also develop analytical abilities and gain exposure to different perspectives, making peer review an effective tool for reflective and cooperative learning [4, 6, 7].

In computing education, peer review takes the form of peer code review. Students evaluate each other's code for quality, correctness, and adherence to best practices [8]. This practice models industry workflows and promotes good programming habits [9, 10]. Peer code review builds on the general benefits of peer review by helping students improve coding skills, deepen their understanding of programming concepts, and enhance collaboration [8, 11–13]. Serving as both reviewer and author also supports reflection, as students learn from evaluating peers' work and receiving feedback on their own code [14].

Despite these benefits, peer review still faces two significant challenges: the provision of low-quality feedback and a lack of alignment with established rubrics. Student comments are often vague, inconsistent, or inaccurate, which limits their usefulness for learning [15–19]. In many cases, students also produce feedback that does not match the rubric, leading to confusion and reducing the effectiveness of the assessment process [20–23]. Addressing these issues is crucial to ensuring that peer review effectively supports the intended learning outcomes, particularly in programming courses. These challenges underscore the need for automated approaches that facilitate more precise rubric alignment during peer code reviews.

Several strategies have been proposed to improve feedback quality, including training programs and online platforms [20, 23, 24]. More recently, AI-based methods, including Machine Learning (ML), Deep Learning (DL), and Large Language Models (LLMs), have been applied to support peer review tasks. These models have been used for text classification, sentiment analysis, and rubric-based evaluation, showing potential for automating the analysis of peer review data [18, 25–28]. However, most prior work focuses on grading, evaluating program correctness, or comparing instructor and peer reviews, rather than classifying feedback according to detailed code quality rubric items in introductory programming. This gap motivates our focus on classifying student feedback specifically by rubric categories.

Few studies have specifically addressed the alignment of rubrics for peer code review in education. Prior research in large programming classes emphasized the consistency of peer reviews [29], while LLM-based work focused on program evaluation rather than student-written feedback [30]. Automated code review models used in software engineering target professional settings and cannot be directly transferred to novice learners [31]. As a result, there is limited work on AI models that classify student peer feedback using detailed rubric criteria for introductory programming. This study fills that gap by adopting the rubric from Stegeman *et al*. [32]. The rubric provides clear criteria for code correctness, readability, structure, and best practices. It includes criteria such as inline comments, header comments, variable names, expressions, layout and formatting, decomposition, modularization, and control flow. This rubric helps guide the design of feedback in different programming contexts.

We present a novel annotated dataset of student peer code review feedback in the Indonesian language, Bahasa Indonesia. The dataset is aligned with the rubric for code quality in introductory programming [32]. To our knowledge, this work is among the earliest attempts to create rubric-annotated resources for peer code review in a low-resource language context. To build interpretable baselines, we

evaluate classical ML models such as Logistic Regression, Support Vector Machine, and Random Forest. These models are efficient and transparent but rely on token-based features, which may overlook important semantic nuances [33]. To capture a richer context, we also test several DL architectures. Convolutional Neural Networks identify local word patterns, while Long Short-Term Memory networks capture sequential dependencies. We further apply Recurrent Convolutional Neural Networks, which combine both approaches for improved context modeling [34]. In addition, we examine few-shot prompting with LLM. This method leverages pre-trained knowledge and works with only a few labeled examples, which is especially useful in low-resource settings [35].

The goal of this study is not only to compare algorithms, but to examine how automated models can support rubric alignment in peer code review. We aim to classify feedback according to rubric criteria that measure code quality. The model is designed exclusively for rubric-category classification. It identifies the type of feedback based on rubric categories and does not perform feedback quality assessment, such as evaluating specificity, usefulness, or constructiveness. We assess how classical ML, DL, and few-shot prompting perform on short and informal feedback written in Bahasa Indonesia. We also analyze which linguistic or structural features lead to misclassifications.

The main research questions are: RQ1: Which model (classical ML, DL, or few-shot) performs best for classifying peer code review feedback according to predefined rubric labels in the introductory programming course delivered in Bahasa Indonesia? and RQ2: What linguistic patterns or contextual overlaps in student peer code review feedback lead to misclassifications by the best-performing models?

This study makes four main contributions. First, it introduces a rubric-annotated dataset of student peer code review feedback in Bahasa Indonesia. Second, it benchmarks three modeling approaches: classical ML, DL, and few-shot prompting. Third, it examines misclassification patterns to identify linguistic and structural issues in student feedback. Fourth, it provides a methodological basis for future tools that can automatically check rubric alignment and support better feedback quality in computing education.

II. Literature Review

A. Machine Learning and Peer Review

ML has effectively addressed several key challenges in student peer review and peer code review, leading to improvements in feedback quality, efficiency, and scalability. One persistent issue, low-quality or unconstructive peer feedback, has been mitigated by AI-supported systems such as EvaluMate, which uses a language model-based chatbot to evaluate and provide real-time feedback on students' review comments. This approach has been shown to enhance the detail, relevance, and overall quality of peer feedback, as students revise their comments in response to AI-generated suggestions [36]. Another major challenge is the labor-intensive process of labeling peer review data for training automated systems. Semi-supervised learning techniques, such as pseudo-labeling, have been introduced to leverage both small labeled and large unlabeled datasets, enabling robust automatic detection of issues in peer reviews while reducing manual labeling effort [37]. Automated classification systems have also been developed to categorize peer review comments into broad types (e.g., praise, problem/solution, verification/summary), helping instructors quickly identify common issues and monitor peer interactions at scale.

ML has also been used to optimize group formation [38, 39] and analyze sentiment or review quality [40]. However, most ML-based peer review studies classify broad feedback types rather than detailed code quality categories.

Recent research demonstrates that a variety of ML methods have been used to analyze student peer review feedback across tasks such as sentiment analysis, topic detection, and rubric-based classification. Traditional models and DL approaches can capture semantic nuances in peer comments and improve classification consistency [40–43]. Yet, these models typically classify broad categories of feedback (e.g., positive/negative, suggestion/problem) and do not map feedback to specific code quality rubric dimensions. This leaves a gap in supporting detailed rubric alignment in programming courses.

B. Student Peer Review Classification

Researchers addressing classification in student peer review predominantly employ ML models to automate the categorization of peer feedback, aiming to support scalable learning analytics and instructional interventions. Most studies treat the classification task as a multi-class (single-label) problem, where each peer review comment is assigned to one exclusive category, such as praise, problem/solution, or verification/summary [38, 44, 45]. However, some research extends this approach by introducing hierarchical or multi-label classification, particularly when comments may simultaneously address multiple aspects (e.g., a problem/solution comment that relates to both writing/formatting and missing content) [38]. This allows for more nuanced analysis, enabling instructors to pinpoint overlapping issues in student writing. Overall, while single-label classification remains the norm for broad feedback categories, multi-label strategies are increasingly adopted for finer-grained subcategories, reflecting the complex and multifaceted nature of peer review comments in educational settings [16, 38, 45].

In both programming and non-programming peer review contexts, several studies have demonstrated how ML approaches can classify reviewers' comments into predefined categories. Ortega *et al*. [18] focus on sentiment analysis, classifying peer feedback as positive or negative, while Huang *et al*. [25] design a system to identify whether students' arguments are relevant to discussion issues. Xiao *et al*. [19] investigate review helpfulness by categorizing textual reviews into attributes such as suggestions, problem identification, and localization. Similarly, Rashid *et al*. [28] extend this work by integrating rubric-based dimensions, such as problem detection and suggestion provision, to identify high-quality feedback. These studies collectively emphasize the adaptability of ML models for handling single-label classification tasks in peer review, while also demonstrating the importance of rubric alignment in guiding classification outcomes.

Researchers have applied a wide range of ML methods to classify student peer review data, addressing both single-label and multi-label classification tasks. Traditional ML algorithms such as Support Vector Machines (SVMs), Decision Trees, Random Forests, Logistic Regression, Naïve Bayes, and k-Nearest Neighbors have been widely used for classifying feedback into distinct categories, with SVMs and Decision Trees often showing strong performance in topic and sentiment classification tasks [41, 43, 46, 47]. More recently, DL approaches such as Convolutional Neural Networks (CNN) and Long Short-Term Memory (LSTM) models have been adopted to capture the nuanced language of student feedback. Ensemble methods, including bagging, boosting, stacking, and voting, have also been explored to improve classification performance, particularly in complex or imbalanced datasets [48]. While most studies focus on single-label classification, assigning each comment to one category, some research has begun to address multi-label scenarios, where feedback may belong to multiple categories simultaneously, leveraging the flexibility of DL and ensemble models to handle such complexity [41, 48]. Despite these advancements, prior studies still focus on general writing or discussion feedback. They do not target peer code review, where comments must match specific code quality rubric categories. This gap remains important in programming education.

C. Peer Code Review Rubrics

Rubrics are widely recognized as essential tools in student peer review, providing clear, structured criteria that guide both assessors and those being assessed. Their use enhances the reliability, validity, and transparency of the assessment process by delineating explicit performance standards and expectations, which helps ensure consistency and fairness in grading across diverse educational contexts [49–52]. Analytic rubrics, which break down tasks into specific components, are particularly effective for formative assessment, supporting detailed feedback and professionalization of student evaluations [49, 50, 52]. Research shows that rubrics not only improve the accuracy of peer and self-assessment but also foster deeper engagement, critical thinking, and self-reflection among students [49, 50, 52–54]. The co-creation of rubrics with students further increases their understanding of assessment criteria and promotes active participation in the evaluation process [55, 56]. Despite these benefits, challenges remain, such as ensuring students' understanding of rubric criteria and addressing potential subjectivity or bias in peer ratings [49, 51, 57].

Rubrics play a crucial role in structuring and enhancing the effectiveness of peer code review in programming courses by providing clear, objective criteria for evaluating code quality. Stegeman *et al*. [32] developed a comprehensive rubric specifically for feedback on code quality in introductory programming courses, iteratively refining it based on an established model of code quality and practical testing across various assignments. Their rubric addresses multiple dimensions of code quality, such as correctness, readability, structure, and adherence to best practices, and offers guidance for drafting new rubrics tailored to different programming contexts [32]. Well-designed rubrics typically address multiple dimensions of code, such as correctness, readability, efficiency, documentation, and adherence to coding standards, which help guide students in both giving and receiving actionable feedback [58, 59]. The research by Indriasari *et al*. [58] demonstrates that using a rubric-based approach, especially when combined with motivational strategies like gamification, can significantly improve the specificity, length, and overall quality of peer feedback, as students are more likely to identify strengths, weaknesses, and offer concrete suggestions for improvement [58, 60]. However, even with well-designed rubrics, students do not always align their comments with the correct rubric criterion. This creates a need for automated support that can classify feedback into specific rubric categories to ensure consistency.

This study focuses on building a single-label classification model for peer code review comments in Bahasa Indonesia. Unlike previous studies that often analyze general feedback or sentiment, this research aims to match comments with specific rubric categories developed by Stegeman *et al*. [32, 61]. The goal is to improve feedback consistency by ensuring that each comment corresponds to a single rubric criterion or a general category. This approach is particularly relevant because some existing peer code review tools use separate text fields for each rubric criterion. For example, the tool proposed by Indriasari *et al*. [58] requires students to enter distinct feedback for each criterion. Thus, automated classification into rubric categories directly supports existing workflow designs for peer code review. Conducting this study in Bahasa Indonesia also addresses the limited resources for research in the peer review area of non-English educational settings.

Given the complexity of text-based peer code review comments, we evaluate a range of supervised ML algorithms, including Random Forest, Support Vector Machine (SVM), and Logistic Regression, to establish interpretable baseline models. However, due to linguistic variability, informal phrasing, and contextual depth, classical ML models may struggle with context-dependent classification [62]. Traditional models treat text as discrete tokens, analyzing words independently and potentially missing semantic nuances [33]. To overcome these limitations, we explore DL architectures such as Convolutional Neural Networks (CNNs) and Long Short-Term Memory Networks (LSTMs), which capture dependencies and context between words. We also utilize Recurrent Convolutional Neural Networks (RCNNs) to enhance classification accuracy further. Sequence-based models like RNNs effectively model word relationships, enabling deeper semantic understanding and better alignment with rubric categories [34]. We also test a few-shot prompting method to handle limited labeled data. By systematically comparing these approaches, this study addresses a clear gap: the lack of automated models that classify peer code review comments into detailed rubric categories, particularly in low-resource language contexts.

III. Materials and Methods

Building on the initial workflow for developing and evaluating classical ML and DL models, this study extends the approach by incorporating few-shot prompting for comparison with supervised methods. Fig. 1 shows the conceptual framework of the system. It illustrates how peer code review feedback in Bahasa Indonesia is converted into

a labeled dataset. The figure also shows how the data is processed using ML, DL, and few-shot prompting models.

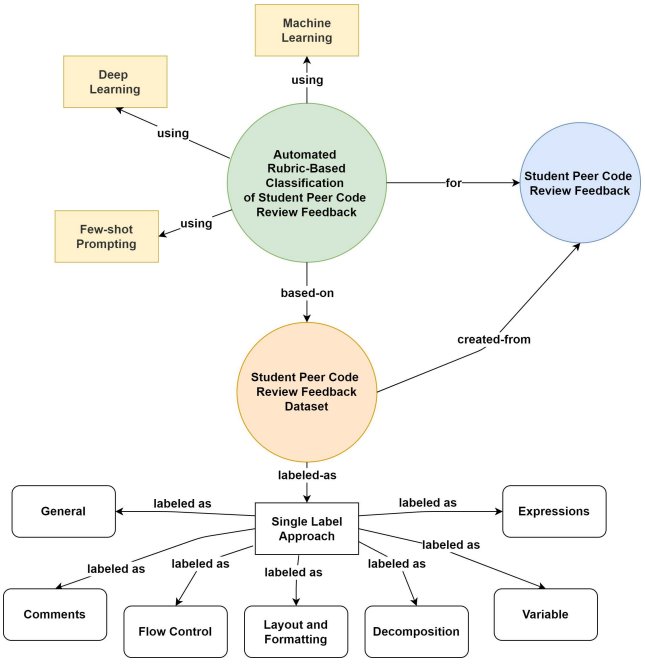


Fig. 1. Conceptual framework of the proposed classification system.

The figure illustrates the main components of the system. It shows how peer code review feedback is collected, converted into labeled data, and processed through ML, DL, and few-shot prompting paths to produce single-label rubric classifications.

This study uses a six-phase workflow to develop and evaluate ML, DL, and few-shot prompting models for classifying peer code review feedback. Table 1 summarizes the workflow, and each phase is described below.

1) Phase 1 (Data Collection): Student peer code review feedback written in Bahasa Indonesia was gathered using a peer code review tool.
2) Phase 2 (Data Preparation): Relevant peer code review data were selected, and experts categorized or corrected labels as needed. Data cleaning was automated.
3) Phase 3 (Feature Engineering): Experts selected feature engineering methods, followed by automated transformation and encoding for models.
4) Phase 4 (Model Training): Experts selected algorithms and data splits with automated model training.
5) Phase 5 (Model Evaluation): Experts chose evaluation metrics, with automated processes assessing model performance.
6) Phase 6 (Few-Shot Prompting Evaluation): Experts chose a few-shot prompting strategy and applied an LLM to classify the feedback in the test set.

For ML baselines, we employed Logistic Regression, Random Forest, and Support Vector Machine, chosen for their interpretability and proven effectiveness in text classification tasks. Features were extracted using Bag-of-Words (BoW) and TF-IDF representations to capture patterns of token frequency. To handle the linguistic variability and contextual depth of Bahasa Indonesia peer feedback, we evaluated DL architectures including Convolutional Neural Networks (CNNs), Long Short-Term Memory Networks (LSTMs), and a Recurrent Convolutional Neural Network (RCNN) hybrid. RCNN was included because it combines CNN's strength in capturing local text features with LSTM's ability to model long-term dependencies. This design aims to improve classification accuracy. These DL models leveraged 300-dimensional FastText embeddings trained on large Bahasa Indonesia corpora to capture syntactic and semantic nuances.

In addition, a separate few-shot prompting experiment was conducted using OpenAI's GPT models (GPT-4.0, GPT-3.5 Turbo, and GPT-4 Mini) in one-shot, three-shot, and five-shot configurations. Unlike ML and DL models that require complete supervised training, few-shot prompting enables classification by conditioning on a small number of annotated examples provided directly in the prompt. This approach is beneficial in low-resource scenarios, such as peer code review feedback in Bahasa Indonesia, where labeled datasets are limited.

Together, the conceptual framework and the six-phase workflow form the complete methodological design of this study.

Table 1. Workflow of the proposed work

Phase	Task	Role
1. Data Collection	Collect feedback from peer code review activities.	Students + peer code review tool
2. Data Preparation	Select relevant peer code review data, manually categorize labels, or revise incorrect labels.	Experts
	Clean data.	Automated Process
3. Feature Engineering	Select features engineering method.	Experts
	Transform and encode features for models.	Automated Process
4. Model Training	Select algorithms and parameters	Experts
	Define train-test split data.	Automated Process
	Train supervised learning models.	Automated Process
5. Model Evaluation	Select evaluation metrics.	Experts
	Evaluate model performance.	Automated Process
6. Few-Shot Prompting Evaluation	Use large pretrained language models (ChatGPT) with one-shot, three-shot, and five-shot prompting for classification; compare results with ML and DL models.	Automated Process + Experts

A. Data Collection

The participants in this study were students from four different classes enrolled in the Object-Oriented Programming course offered by the Department of Informatics at a university in Indonesia. In total, 171 students took part in the course, with the vast majority (98.25%) being second-year students. Several processes were conducted to collect and prepare data. Peer code review activities were prepared as part of an Object-Oriented Programming (OOP) course taught in Java. The topics covered included objects, classes, abstraction, encapsulation, inheritance, polymorphism, and design principles.

Students participated in a peer code review assignment, for which they were provided with a document explaining the peer code review process. The document emphasized the importance of accurate scoring and detailed feedback, focusing on both strengths and weaknesses, while also providing suggestions for improvement. Each student submitted a program completed during a designated laboratory session. The submissions were distributed anonymously using a peer code review tool, and each student was randomly assigned to review at least five peer-submitted programs within a seven-day period. In total, 137 students submitted their code, and 115 participated in the code review activity.

Students applied a consistent set of rubric criteria for assessing code quality based on those outlined by Stegeman *et al*. [32, 61], covering topics such as Inline Comments, Header Comments, Variable Names, Expression, Layout, Formatting, Decomposition, Modularization, and Control Flow. To streamline the evaluation process, similar criteria were consolidated: Inline Comments and Header Comments were merged into a single "Comments" criterion, Layout and Formatting were combined into a "Layout and Formatting" criterion, and Decomposition and Modularization were unified into a "Decomposition" criterion. This resulted in six criteria for code quality assessment during the peer code review assignment. While students identified defects and discussed them in the feedback sections, these criteria primarily focused on code readability and modularity. Each student anonymously assessed their peers' code using a numeric scale (1–5) for each criterion, with six criterion-specific text boxes for written feedback. The students' written feedback from each text box was used as a corpus in natural language processing.

This study was approved by the University of Auckland Human Participants Ethics Committee (Approval No. 024478). Informed consent was obtained from all individual participants included in the study.

B. Data Preparation

The preprocessing steps involved loading the dataset from an Excel file, filtering out missing values in the feedback column, and removing duplicate entries while retaining the last occurrence. After cleaning, the dataset was reduced from 4385 to 2281.

Table 2. Examples of manually categorized feedback and definition of each label [32]

Student Feedback (Original language)	Student Feedback (Translation to English)	Label	Definition
Penamaan variabel sudah cukup baik dan jelas. Setiap variabel bisa dibaca dengan jelas sehingga bisa mengetahui apa fungsi dari tiap variabel. Mohon dipertahankan.	The variable naming is quite good and clear. Each variable is easily readable, making it possible to understand the function of each variable. Please maintain this.	Variable	Feedback that focuses on the naming of variables, including whether variable names are meaningful, descriptive, and clearly represent their purpose or the data they store. This label does not address how variables are used in expressions or computations.
Expressions pada program ini sudah sesuai, karena telah mrnggunakan rumus yang sederhana dan menggunakan tipe data yang tepat.	The expressions in this program are appropriate because they use simple formulas and the correct data types.	Expressions	Feedback that focuses on the use of expressions, formulas, operators, and data types in the code, including simplicity, correctness, and appropriateness. This label addresses how values are computed or combined, not how variables are named.
Control flow menghasilkan keluaran yang sesuai ekspektasi, sederhana, exception kurang dihandle dengan baik, nah masih ada beberapa inputan yang bisa ditembus atau tidak mengeksekusi semantic error. Saran: bisa ditambahkan exception handling seperti menggunakan try catch.	The control flow produces outputs that meet expectations and is straightforward. However, exceptions are not handled well, and there are still some inputs that can bypass or fail to execute due to semantic errors. Suggestion: Add exception handling, such as using try-catch.	Control Flow	Feedback that focuses on the structure and logic of program execution, including conditional statements, loops, branching logic, and exception handling, as well as the clarity of execution flow.
Sudah ada penggunaan comments baik pada header maupun inline. Tetapi saya hanya menemukan comment pada beberapa bagian saja, bagian yang selalu diberi comment adalah pada penggunaan super. Sisanya, hanya beberapa method yang terdapat comments.	Comments have been used both in the header and inline. However, I only found comments in a few sections; the sections that consistently have comments are related to the use of "super." The rest only include comments in a few methods.	Comments	Feedback that evaluates the presence, clarity, completeness, and usefulness of comments, including header comments and inline comments within the code.
Layout sudah cukup baik dengan indentations yang cukup konsisten dari awal hingga akhir. Tetapi sebaiknya menggunakan spasi dengan cukup baik, mengelompokkan code sesuai fungsi serta menggunakan enter untuk memisahkan code yang berbeda.	The layout is quite good, with consistent indentations from start to finish. However, it would be better to use spacing properly, group the code based on its function, and use line breaks to separate different sections of code.	Layout and Formatting	Feedback that focuses on code readability and visual structure, including indentation, spacing, grouping of code blocks, blank lines, and consistent use of brackets or braces.
Bagian dekomposisi sudah baik karena tugas sudah dibagi dalam kelas masing masing dan tujuannya spesifik	The decomposition is good because tasks have been divided into specific classes, each with a clear purpose.	Decompositi on	Feedback that focuses on how well the code is divided into functions, methods, classes, or modules, with clear responsibilities, minimal duplication, and limited shared variables.
Overall sudah sangat baik, tapi masih bisa dikembangkan lagi	Overall, it is very good but still has room for further improvement.	General	Feedback that does not clearly fit into any specific rubric category, including overall impressions, general praise, or remarks related to assignment requirements or submitted files.

We applied a consistent text cleaning pipeline to reduce noise while preserving the meaning of student feedback. First, we converted fully uppercase tokens into title case to reduce unnecessary casing variation. We then removed URLs and HTML tags using regular expressions. We also removed emojis and pictographic symbols using a Unicode-based filter.

We performed stop-word removal using a custom Indonesian stop-word list. The list was loaded from a text file and applied to each feedback. Tokenization was conducted at the word level using the text_to_word_sequence function. After tokenization, tokens that appeared in the stop-word list were removed, and the remaining tokens were joined back into a cleaned sentence. If a feedback became empty after stop-word removal, it was removed from the dataset to avoid invalid inputs during training.

In this study, a dataset of 2,281 peer code review feedback items was manually annotated based on six predefined rubric criteria: Variable, Expression, Control Flow, Comments, Layout and Formatting, and Decomposition, with an additional General label for feedback that did not fit any of the other criteria. Two experts collaboratively performed the annotation following predefined rules. Expert 1 labeled the entire dataset, while Expert 2 independently annotated a randomly selected subset of 350 feedback covering all labels. This subset size was determined using the Slovin formula to ensure statistical validity. The dual annotation process was designed to verify the consistency and reliability of the labeling. Table 2 provides examples and definitions of feedback in each label in both the original language and its translations.

Ambiguous feedback was addressed through the use of an annotation guideline and a discussion process. If the feedback could fit more than one rubric label, the annotators assigned the label that best represented the main intent of the comment. If no label was clearly applied, the comment was assigned to the General category.

To measure the agreement between the two experts, Cohen's Kappa was calculated, yielding a high inter-rater reliability of 0.9463. Cohen's Kappa was specifically chosen due to its suitability for inteSr-rater reliability between exactly two raters annotating categorical data, especially when the second rater annotates only a subset of the entire dataset. This high kappa value (0.9463) indicates near-perfect agreement between the annotators [63], confirming the reliability and quality of the annotation procedure used in this study.

C. Feature Engineering

Our study is organized into two experimental tracks: classical ML algorithms and DL models. This design allows us to evaluate feature effects within ML and DL paradigms that explicitly rely on feature engineering, while also benchmarking them against prompting-based approaches that require minimal feature preparation.

1) ML pipeline

In the ML experiment, Logistic Regression, Random Forest, and Support Vector Machines receive only Bag-of-Words (BoW) inputs and TF-IDF with raw count vectors computed for single words, 2–3-word phrases, and character n-grams, providing a surface-level benchmark based solely on term frequency. An automated preprocessing pipeline handles tokenisation and constructs the BoW and TF-IDF representations.

2) DL pipeline

In the separate DL experiment, the same tokenised corpus is fed to a neural model whose input layer is a 300-dimensional FastText embedding matrix trained on large Bahasa Indonesia corpora. FastText, as shown by Bojanowski *et al*. [64], represents each word as the sum of its character-n-gram vectors; this subword composition implicitly encodes syntactic patterns (e.g., affix cues to part of speech) and semantic relatedness (shared stems and morphemes), outperforming Word2Vec and GloVe on analogy tasks in morphologically rich languages. A separate stage in the pipeline performs FastText embedding lookup, ensuring that any performance differences reflect the richer syntactic–semantic information encoded in FastText rather than confounding factors.

D. Model Training

This study conducts a series of experiments employing ML and DL to classify peer code review feedback according to predefined rubric criteria. The models classify only the type of feedback based on rubric categories and do not assess the quality of the feedback, such as its specificity or constructiveness.

1) ML pipeline

The ML experiments for Logistic Regression, Random Forest, and Support Vector Machines were conducted using a unified training and evaluation pipeline. This procedure begins with k-fold cross-validation on the training data to obtain stable estimates of model performance, where each classifier is trained and validated across multiple folds using weighted multi-class scoring. After cross-validation, the model is retrained on the whole training set to maximize its exposure to all available labeled examples. The final trained model is then evaluated on a separate held-out test set, ensuring that the reported results reflect true generalization to unseen data. The pipeline also records training and prediction times and can optionally save the final fitted model, providing a consistent, rigorous, and reproducible framework for comparing ML algorithms.

2) DL pipeline

The DL experiments follow a structured training and validation pipeline that mirrors the evaluation logic used in ML models while accommodating the requirements of neural network training. The procedure begins with stratified k-fold cross-validation, where a fresh copy of the neural network is instantiated and trained on each fold using early stopping to prevent overfitting. For every fold, the model is trained on the training split and validated on the held-out fold, and a suite of performance metrics is computed using weighted scoring to account for class imbalance. After completing all folds, the neural network is reinitialized and trained on the full training dataset with a validation split to ensure stable optimization. The final trained model is subsequently evaluated on the held-out test set, which was never used during cross-validation. This workflow enables DL models to benefit from robust fold-level performance estimation, ensuring that the final reported metrics accurately reflect generalization

performance on unseen data. The pipeline also records training and inference times. It optionally saves the best-performing model, providing a comprehensive and reproducible evaluation framework for all neural network architectures used in the study. The illustration of the DL architecture is shown in Fig. 2.

To ensure a clear separation between model tuning and final testing, we applied a two-stage validation process: a hold-out split followed by cross-validation within the training set. First, the dataset of 2281 annotated feedback was divided by label and split into 80% for training (1825 feedback) and 20% for testing (456 feedback). Stratification maintained the class distribution in both sets to avoid imbalance issues. Next, the training set underwent stratified 5-fold cross-validation, with each fold containing approximately 365 feedback. The average validation scores were used to select the best model settings. After tuning, the model was retrained on the entire training set and evaluated once on the hold-out test set to measure its generalization ability. For the DL models, early stopping was applied to prevent overfitting by monitoring the training loss, and a Softmax layer was used in the output layer to assign associated labels.

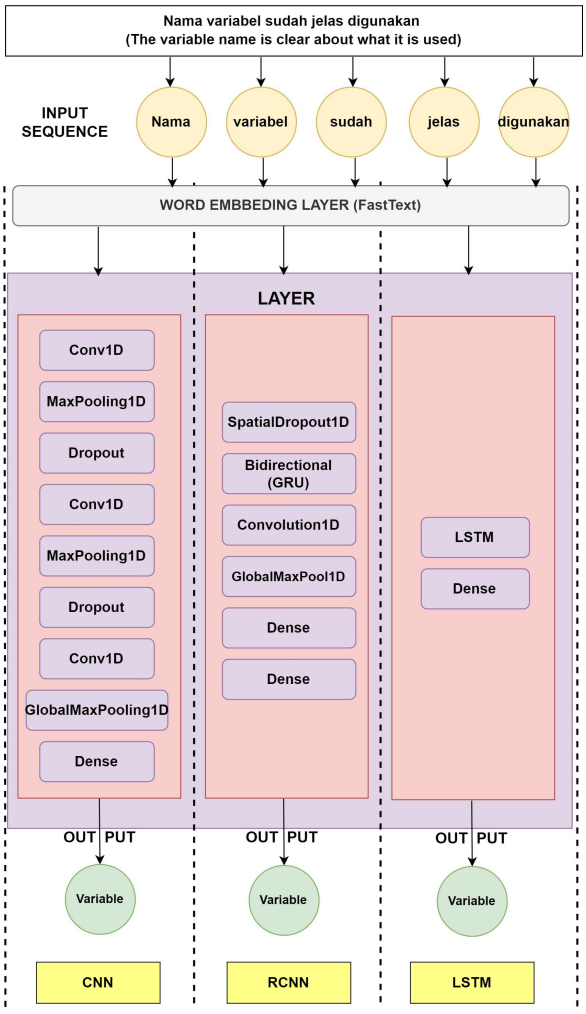


Fig. 2. DL architecture for peer code review classification.

While Fig. 2 illustrates the conceptual deep learning architectures, this study employs a concrete RCNN configuration with fixed hyperparameters. The RCNN begins with a word embedding layer using 300-dimensional FastText vectors loaded as a fixed (non-trainable) embedding matrix. To regularize the embedding output, SpatialDropout1D with a dropout rate of 0.30 is applied. Contextual representations are captured using a Bidirectional GRU with 32 hidden units and a tanh activation function. The sequence output is then processed by a 1D convolution layer with 32 filters and a kernel size of 3, followed by global max pooling. The resulting feature vector is passed to a Dense layer with 25 units and ReLU activation, and finally to a Softmax output layer for multi-class classification.

E. Model Evaluation

The evaluation of the ML, DL, and few-shot prompting models was conducted using standard classification metrics, with a primary focus on the F1-score to ensure a comprehensive assessment of performance. In text classification, each prediction falls into one of four categories: True Positive (TP), representing correctly predicted positive instances; True Negative (TN), representing correctly predicted negative instances; False Positive (FP), where a negative instance is incorrectly classified as positive; and False Negative (FN), where a positive instance is incorrectly classified as negative. Precision, recall, and F1-score were computed using the standard formulations shown below (Eq. (1), Eq. (2), and Eq. (3)), which follow established principles for multi-class evaluation [65].

$$Recall = \frac{TP}{TP+FN} \in [0,1] \quad (1)$$

$$Precision = \frac{TP}{TP+FP} \in [0,1] \quad (2)$$

$$F1 = 2 \times \frac{Precision \times Recall}{Precision+Recall} \in [0,1] \quad (3)$$

Final predictions from the best-performing ML and DL models were generated using a separate test dataset, distinct from the data used during training and cross-validation. This independent dataset served as the basis for both quantitative performance evaluation and qualitative error analysis.

To support RQ2, per-label metrics and confusion matrices were constructed to identify systematic misclassification patterns and highlight categories that the models found challenging to distinguish. Additionally, word clouds were generated for each class in the test set to visually illustrate dominant lexical features that may influence classification behavior and contribute to model errors.

F. Few-Shot Prompting Evaluation

The few-shot prompting experiments were designed to evaluate the capability of LLMs to classify peer feedback without relying on the preprocessing pipeline used in the ML and DL approaches. Unlike traditional models, LLMs can operate directly on raw text when guided by a carefully structured prompt.

Each prompt follows a fixed and consistent structure consisting of four main components:

1) an explicit task instruction requiring the model to assign exactly one rubric label to each feedback instance;
2) detailed definitions of the seven rubric categories used in this study;
3) strict output constraints specifying the required CSV format and prohibiting additional explanations; and
4) a set of labeled examples illustrating the expected

classification behavior.

To examine the effect of example-based conditioning, three prompting configurations were evaluated: one-shot, three-shot, and five-shot prompting. In the one-shot configuration, a single labeled feedback example is provided after the rubric definitions. In the three-shot and five-shot configurations, the same prompt template is extended by including three or five labeled examples, respectively, covering representative rubric categories. Apart from the number of examples, all prompt components and instructions remain identical across configurations to ensure methodological consistency. The complete prompt templates, including task instructions, rubric definitions, output rules, and illustrative examples for each shot configuration, are provided in the Appendix to support full reproducibility.

The few-shot strategies were evaluated using several LLM variants from the GPT model family, including GPT-4.o, GPT-3.5 Turbo, and GPT-4 Mini (o4-mini). Each model received the same prompt template, ensuring consistency across conditions. For evaluation, the complete test dataset was submitted to each of the few-shot configurations using the constructed prompt.

The resulting model-generated predictions were collected and analyzed using the same classification metrics and procedures applied to the ML and DL models. This ensured methodological comparability across all approaches. Performance was assessed using precision, recall, and F1-score, complemented by confusion matrix analysis to identify misclassification patterns across rubric categories.

To further support transparency and reproducibility, upon acceptance of the paper, the annotated dataset and the complete source code used in this study will be made publicly available.

IV. Results and Discussions

The dataset, after preprocessing, contained 2,281 feedback items across seven labels, as shown in Fig. 3. The most frequent labels were Comments (354), Layout and Formatting (351), and Flow Control (350), followed by Variable (348), Decomposition (319), Expressions (294), and General (265). The computed class weights reflect this distribution: Flow Control (0.902), Layout and Formatting (0.910), Comments (0.956), Variable (0.943), Decomposition (1.053), Expressions (1.103), and General (1.205), with lower weights for more frequent labels and higher weights for less frequent ones. The dataset was determined to be balanced (ratio = 0.747), indicating that class frequencies are relatively proportional.

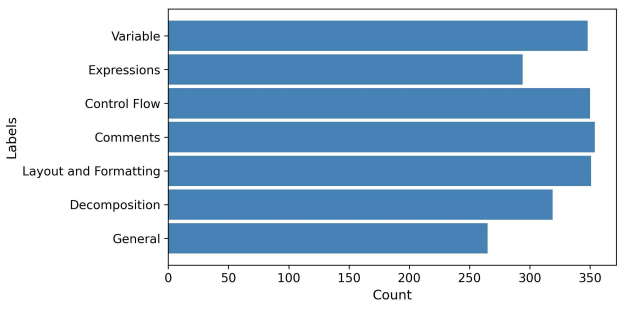


Fig. 3. Distribution of labels in the dataset.

A. RQ1: Which Model (Classical ML, DL, or Few-Shot) Performs Best for Classifying Peer Code Review Feedback According to Predefined Rubric Labels in the Introductory Programming Course Delivered in Bahasa Indonesia?

Given that our corpus of 2,281 peer code review feedback in Bahasa Indonesia is novel and represents the dataset of its kind explicitly annotated with rubric-based categories, no direct SOTA model existed prior to this study. Consequently, we developed our baseline model by systematically evaluating classical ML models. This baseline was compared with advanced DL architectures (CNN, LSTM, RCNN) as well as an SOTA LLM (GPT-3.5 Turbo) through few-shot prompting techniques, ensuring consistent evaluation criteria across all methods.

1) ML performance

The first experiment evaluated multiple ML classifiers, including Logistic Regression, Random Forest, and Support Vector Machine, using different feature engineering techniques such as Count Vectorization, Word-Level TF-IDF, N-Gram TF-IDF, and Character-Level TF-IDF, as seen in Table 3. Overall, the findings suggest that both the type of classifier and the representation of features play a significant role in classification performance, as indicated by the overall F1-score obtained through cross-validation. Among the three algorithms tested: Logistic Regression, Random Forest, and Support Vector Machine, the basic Count Vector method clearly performs better than the more complex N-gram TF-IDF features (for example, 0.9387 vs. 0.8499 in Logistic Regression, 0.9430 vs. 0.8222 in Random Forest, and 0.9171 vs. 0.8540 in SVM). Thus, in our task, basic token counts separate the classes more clearly than advanced *n*-gram features. On the other hand, the extra weighting and sparsity from using TF-IDF with words or *n*-grams may weaken those signals instead of improving them.

Table 3. Performance metrics of ML classifiers with various feature engineering techniques

Classifier	Feature Engineering	F1-score
Logistic Regression	Count Vector	0.9387
	Word Level TF-IDF	0.9250
	Ngram TF-IDF	0.8499
	Character Level TF-IDF	0.9378
Random Forest	Count Vector	0.9430
	Word Level TF-IDF	0.9164
	Ngram TF-IDF	0.8222
	Character Level TF-IDF	0.9352
Support Vector Machine	Count Vector	0.9171
	Word Level TF-IDF	0.9243
	Ngram TF-IDF	0.8540
	Character Level TF-IDF	0.9409

Character-level TF-IDF also gives excellent results, coming close to or slightly below the performance of the Count Vector method. It reaches F1-scores of 0.9378 and 0.9409 for Logistic Regression and SVM, and 0.9352 for Random Forest. This indicates that patterns at the sub-token level (such as variable names, code fragments, or punctuation marks) are also beneficial for this task. Representing text at the character level can catch minor spelling or style details that word-based methods might miss. When comparing all classifiers, Random Forest with Count Vector yields the best F1-score of 0.9430, slightly outperforming Logistic Regression with Count Vector (0.9387) and SVM with

Character TF-IDF (0.9409). Random Forest also handles weaker feature types well: even with Word-Level TF-IDF, it still scores 0.9164, which is better than Logistic Regression (0.9250) and SVM (0.9243) using the same features. Because of its strong and stable performance across different feature types, Random Forest with Count Vector (F1 = 0.9430) is chosen as the baseline model for this peer code review classification task.

2) DL performance

The second experiment, which involves DL, shows that each model's performance depends on how its architecture interacts with the detailed language features captured by FastText embeddings (see Table 4). The basic CNN model achieves an F1-score of 0.5439, indicating that although FastText captures parts of words, a simple convolutional layer struggles to fully comprehend the range of peer code review feedback submitted by students and eliminate unhelpful patterns. When using LSTM, the F1-score increases sharply to 0.8322. This improvement shows that FastText is effective at representing parts of words, such as code-related prefixes, suffixes, and common spelling errors. These detailed representations work well with the LSTM's ability to understand sequences in the text. The LSTM can follow the flow of a review feedback, using FastText's subword features to keep a consistent understanding of unusual words and code terms.

Table 4. Performance metrics of DL classifiers with FastText embeddings

Classifier	Model	F1-score
Convolutional Neural Networks (CNN)	CNN_WE	0.5439
Long Short-Term Memory (LSTM)	LSTM_WE	0.8322
Region-based Convolutional Neural Networks (R-CNN)	RCNN_WE	0.9113

The Region-based CNN achieves the highest F1-score of 0.9113, making it the most suitable DL model for this task. This model combines region-based convolution to detect important phrase patterns with recurrent pooling to integrate those features into a comprehensive understanding. FastText embeddings help throughout by capturing word structure through character-level *n*-grams and showing meaning by placing similar technical terms close together in the embedding space. This combination enables RCNN to achieve the highest F1-score overall, making it a strong reference point for future improvements.

3) Few-shot prompting comparison

The third experiment, which uses few-shot prompting, demonstrates that model performance varies based on both the model architecture and the number of examples given, as shown in Table 5. GPT-4.0 improves with more examples, up to the three-shot setting (F1 = 0.746), but its performance drops slightly with five shots (F1 = 0.664). GPT-3.5 Turbo shows the highest overall F1-score of 0.852 in the five-shot setting, indicating it benefits most from additional context. In contrast, GPT-4 Mini (o4-mini) performs best with just one example (F1 = 0.787), but its performance declines as more examples are added. These findings suggest that GPT-3.5 Turbo is more capable of leveraging multiple examples effectively, while GPT-4 Mini is more efficient in lower-shot scenarios.

Table 5. Performance metrics of few-shot classifiers

Model	Few-shot	F1-score
GPT-4.o	One-shot	0.613
	Three-shot	0.746
	Five-shot	0.664
GPT-3.5 Turbo	One-shot	0.747
	Three-shot	0.685
	Five-shot	**0.852**
GPT-4 Mini (o4-mini)	One-shot	0.787
	Three-shot	0.679
	Five-shot	0.620

In summary, while RCNN shows solid performance for a DL model and few-shot prompting provides promising results, Random Forest with Count Vector features remains the most effective and reliable baseline. Its ability to directly leverage token-level patterns, manage sparse input well, and provide fast, interpretable outcomes makes it a strong foundation for future enhancements.

B. RQ2: What Linguistic Patterns or Contextual Overlaps in Student Peer Code Review Feedback Lead to Misclassifications by the Best Performing Models?

After discussing the model development, this section presents an analysis of the best model performances for ML, DL, and few-shot experiments. First, we conducted word cloud analysis for seven labels from the test dataset. Second, we perform the error analysis using the confusion matrix.

In addition to the model's development, we also conducted word cloud analysis for seven labels. We used the dataset for model testing to ensure that the insights remain generalizable and reflective of real-world student peer code review behavior, rather than being influenced by the patterns the model has already learned. The word clouds visualize the most frequently used words in peer code review feedback (see Fig. 4a to Fig. 4g), helping to identify common themes in students' feedback.

1) Word cloud analysis

a) Variables (see Fig. 4a)

Feedback focuses on naming clarity and adherence to conventions, often referencing variable *("variabel")*, variable naming *("penamaan variabel")*, variable name *("nama variabel")*, and whether names are clear *("jelas")* or appropriate *("sesuai", "baik")*. These patterns indicate that students perceive variable naming as crucial to code readability and clarity of intent.

b) Expressions (see Fig. 4b)

Feedback patterns relate to the correctness and appropriateness of expressions, including the use of *"expression",* formula *("rumus")*, and data type *("tipe data")*. Students highlight whether these elements are appropriate *("sesuai", "sesuai kebutuhan")* in relation to program needs and easy to understand ("mudah dipahami"), demonstrating attention to both logical accuracy and readability.

c) Control flow (see Fig. 4c)

Feedback in this category highlights students' attention to control structures and *"exception handling"*. Frequent terms

such as *"exception", "control flow", "program", "baik"* (good), and *"sesuai ekspektasi" (*meets expectations*)* indicate that reviewers evaluate how well control flow accommodates different scenarios and whether edge cases are appropriately addressed.

d) Comments (see Fig. 4d)

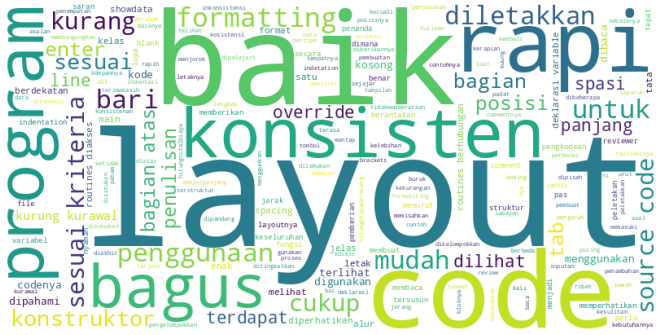
Feedback in this category centers on explanations within the code, particularly the placement and clarity of *"comment"*, *"header"*, and *"inline"* descriptions. Students often point out missing or unclear explanations of function *("fungsi")* or program logic, as reflected in terms like "lacking" ("kurang") and "should" ("sebaiknya"), indicating an expectation that comments should enhance readability.

(a)

(b)

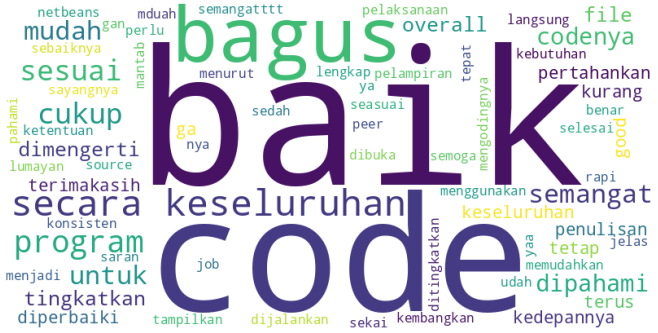
(c)

(d)



(e)

(f)



(g)

Fig. 4. Word clouds for the seven rubric labels: (a) Variable, (b) Expressions, (c) Control Flow, (d) Comments (e) Layout and Formatting, (f) Decomposition, and (g) General.

e) Layout and formatting (see Fig. 4e)

Reviews consistently emphasize readability through *"layout"*, spacing *("spasi")*, neat *("rapi")*, and consistent formatting *("konsisten")*. Students value visually organized code *("baik", "bagus")* and often note when alignment or spacing is inappropriate *("kurang")*, suggesting strong expectations for formatting standards.

f) Decomposition (see Fig. 4f)

Reviews emphasize the structuring of code into coherent units, such as classes ("kelas"), modules ("modul"), and logical divisions *("pembagian")* of tasks *("tugas")*. The mix of positive cues like good *("baik)* and critical terms such as suboptimal ("kurang optimal") reflects students' focus on whether decomposition is sufficiently modular and functionally separated.

g) General (see Fig. 4g)

This category captures broad assessments of overall code quality, using terms such as good, great ("*baik"*, "*bagus")*, "*code"*, and overall *("secara keseluruhan")*. The feedback tends to be evaluative rather than specific, reflecting general impressions rather than a detailed critique.

2) Confusion matrix analysis

The confusion matrix in Fig. 5 shows that the Random

Forest model achieves high accuracy across all seven labels, with strong True Positive counts for Comments, Expressions, Variable, and Decomposition. The remaining errors are limited and mostly occur between labels that share similar linguistic features. For example, some instances of Layout and Formatting and Expressions were misclassified as General, while Flow Control was occasionally confused with Expressions or General. One student's feedback illustrates this overlap:

"Project tugas praktikum kedua miliki terlihat baik penanganan user memasukkan menu 0 4 diminta memasukkan menu ulang sayangnya pengisian data tipe data integer dimasukkan huruf tertampilkan diminta menginputkan kembali bagian salah." (original)

"When the user enters menu option 0 or 4, they are prompted to re-enter the menu. Unfortunately, when a letter is entered instead of an integer data type, the input is displayed and the user is asked to re-enter the incorrect part." (English translation)

This example was annotated as Flow Control but predicted as Expression. It is likely because the text includes terms related to data types, which can also appear in the Expression label.

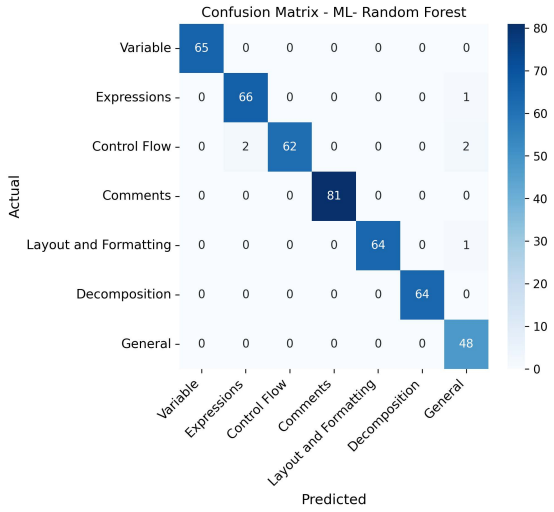


Fig. 5. Confusion matrix of the best-performing ML model (random forest with count vector features).

Table 6. Precision, recall, and F1-score for the ML best model (random forest)

Label	Precision	Recall	F1-score	Support
Variable	1.00	1.00	1.00	65
Expressions	0.97	0.99	0.98	67
Flow Control	1.00	0.94	0.97	66
Comments	1.00	1.00	1.00	81
Layout and Formatting	1.00	0.98	0.99	65
Decomposition	1.00	1.00	1.00	64
General	0.92	1.00	0.96	48

Table 6 shows that the Random Forest model performs strongly across all seven labels. Variable, Comments, and Decomposition achieve perfect precision, recall, and F1 Scores, indicating a very reliable classification. Layout and Formatting, as well as the Expressions label, also show high performance, with F1-scores of 0.99 and 0.98. Flow Control has a slightly lower recall of 0.94, which matches the confusion patterns noted earlier. The general category has the lowest precision at 0.92, suggesting some overlap with linguistically similar categories. Overall, the metrics confirm the model's robustness and are consistent with the analysis of the confusion matrix.

The confusion matrix in Fig. 6 indicates that the RCNN model produces more misclassifications than the Random Forest model, particularly for Layout and Formatting and Flow Control. Layout and Formatting records 58 true positives but is misclassified several times into General, Flow Control, and Decomposition, suggesting that the model struggles to separate formatting-related comments from broader or structural feedback. Flow Control shows a similar pattern, with 59 correct predictions but errors across General, Expressions, Comments, and Decomposition. These mistakes reflect difficulties in capturing subtle contextual cues. One example of this issue is the following feedback, which belongs to Flow Control but was classified as General:

"Exception sudah ada namun belum baik" (original)

"Exception handling is already present but not well implemented." (english translation)

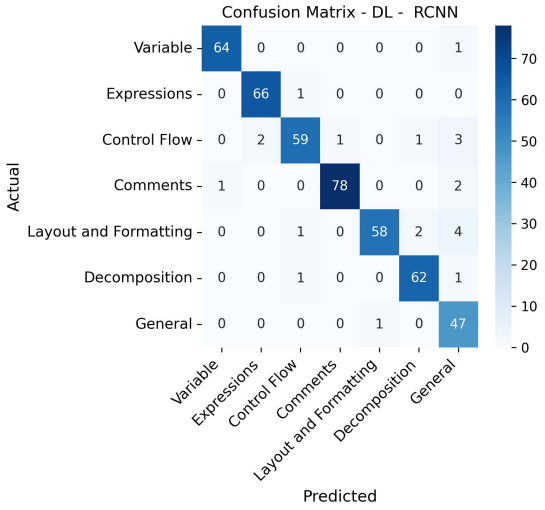


Fig. 6. Confusion matrix of the best-performing DL model (RCNN with word embeddings).

Table 7 supports the confusion patterns observed in Fig. 6. The RCNN model shows strong performance for most labels, with high F1-scores for Expressions (0.98), Comments (0.97), and Decomposition (0.96). Variable also performs well with an F1-score of 0.98. However, the Control Flow and Layout and Formatting records have lower recall values of 0.89, which is consistent with the misclassification trends discussed earlier. The general label has the lowest precision at 0.81, indicating that several unrelated comments were assigned to this label. This suggests that RCNN has more difficulty distinguishing labels with overlapping textual features.

Table 7. Precision, Recall, and F1-score for the DL Best Model (RCNN)

Label	Precision	Recall	F1-score	Support
Variable	0.98	0.98	0.98	65
Expressions	0.97	0.99	0.98	67
Control Flow	0.95	0.89	0.92	66
Comments	0.99	0.96	0.97	81
Layout and Formatting	0.98	0.89	0.94	65
Decomposition	0.95	0.97	0.96	64
General	0.81	0.98	0.89	48

Fig. 7 displays the confusion matrix for the optimal five-shot prompting setup utilizing GPT-3.5 Turbo. The model achieves high true positives across all seven labels, indicating effective recognition of key terms and patterns. Most errors involve assigning specific labels to the broader General category. For example, feedback about control flow was misclassified as General:

"code ini menghasilkan keluaran yang sudah sesuai ... tetapi logika penentuan durasi proses bisa diganti dengan 'lama kerja' agar tidak membingungkan user." (original)

"This code produces the correct output ... but the logic for determining the process duration could be changed to 'work duration' to avoid confusing the user." (English translation)

The intended label was Flow Control, but the model treated the feedback as general advice. Similar errors occurred when feedback about layout appeared too general to the model.

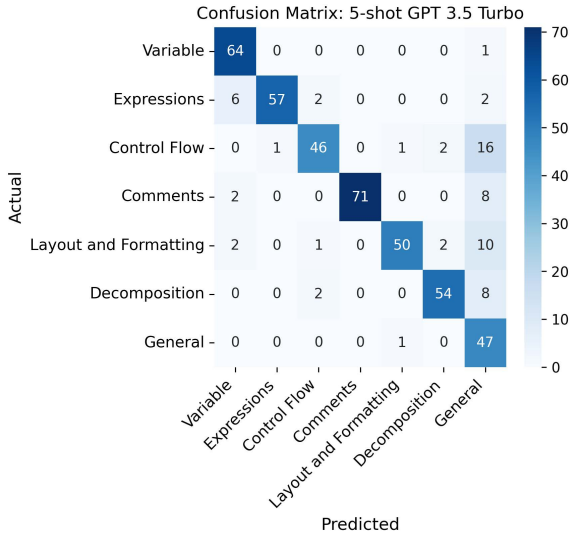


Fig. 7. Confusion matrix of the best-performing few-shot prompting model (5-shot GPT-3.5 turbo).

Table 8 reinforces the misclassification patterns seen in Fig. 7. The five-shot GPT-3.5 Turbo model performs well on several labels, with high F1-scores for Comments (0.93), Variable (0.92), and Expressions (0.91). However, Control Flow and Layout and Formatting show lower recall values of 0.70 and 0.77, indicating that many instances were overlooked or incorrectly assigned to other categories. Decomposition also shows moderate performance with an F1-score of 0.88. The General label yields the weakest precision at 0.51, which aligns with the confusion matrix, which shows frequent misassignments from more specific labels. Overall, these results reflect the model's tendency to interpret detailed feedback as general advice, especially when the text contains broad or ambiguous phrasing.

Table 8. Precision, recall, and F1-score for the few-shot best model (5-shot GPT-3.5 turbo)

Label	Precision	Recall	F1-score	Support
Variable	0.86	0.98	0.92	65
Expressions	0.98	0.85	0.91	67
Control Flow	0.90	0.70	0.79	66
Comments	1.00	0.88	0.93	81
Layout and Formatting	0.96	0.77	0.85	65
Decomposition	0.93	0.84	0.88	64
General	0.51	0.98	0.67	48

The analysis of RQ2 reveals that misclassifications frequently occur when peer code review feedback contains overlapping wording or context between labels, resulting in some feedback, such as on Flow Control, Expressions, and Layout, being classified as general feedback. Overall, while the ML, DL, and few-shot models perform well, subtle similarities in language and context make it harder to distinguish certain labels accurately.

C. Discussion and Future Work

This study explored the classification of peer code review feedback using ML, DL, and few-shot prompting approaches in computing education, demonstrating their effectiveness in aligning feedback with predefined rubric criteria. The best-performing ML model, Random Forest with Count Vectorization, achieved an F1-score of 0.9430, outperforming the more advanced RCNN model with FastText embeddings (F1 = 0.9113). This finding highlights that simpler models can outperform complex ones when the task and data characteristics are well-suited to their design.

Rubric-based peer code review feedback often contains explicit tokens, such as function names, variable references, and error codes. Token count features can effectively capture these elements. Random Forest's ensemble approach leverages these tokens, identifying even rare but meaningful patterns without losing information during processing. Similar findings have been reported in prior research, where Random Forest demonstrated strong performance in text classification tasks and, in some instances, surpassed more complex approaches [66].

Despite the theoretical advantage of DL in capturing contextual relationships between words [67]. Deep models, such as RCNN, generally perform best with large, well-labeled datasets that enable them to learn subtle patterns and semantic relationships. With limited data, these models may lose rare but important tokens due to multiple processing layers. Random Forest, by contrast, directly captures these signals and incorporates built-in regularization that helps prevent overfitting. DL models require careful tuning and are more prone to overfitting on small patterns or noise. Our results suggest that, for this task, the stability and robustness of an ensemble method like Random Forest are more effective than those of a deeper RCNN architecture.

The few-shot prompting experiment further reinforces these findings. While GPT-based models showed promising results, they did not outperform Random Forest. This suggests that, despite advancements in prompting-based methods, classical ML with well-matched feature engineering remains a strong and consistent solution for rubric-based peer code review classification.

Furthermore, Random Forest offers practical advantages in speed, interpretability, and deployment. It trains significantly faster than DL models, enabling rapid experimentation and refinement, and provides clearer insights into which tokens influence classification outcomes. These practical strengths make Random Forest particularly suitable for integration into peer code review tools aimed at improving feedback quality in programming education.

Model interpretability is also essential for educators. They need to know why the model assigns the student feedback to a specific label. Random Forest helps with this because it

provides feature-importance scores for tokens or n-grams. Educators can check which words influence each label, such as terms about variable naming, layout, or exception handling. This helps them see whether the model is using rubric-based cues or noise. Frameworks such as LIME [68] and SHAP [69] can also show which parts of a student's feedback contribute most to the prediction. These frameworks make the classification process more transparent and easier for instructors to understand.

Producing high-quality written feedback is essential in student peer review activities, benefiting both reviewers and authors [58, 70, 71]. Reviewers can enhance their evaluation skills, while authors can receive targeted, constructive suggestions for improvement. Prior studies indicate that well-structured feedback enhances assessment clarity and accuracy while also fostering deeper engagement with evaluation criteria, leading to better learning outcomes [71–76].

These findings can help improve the quality and consistency of peer reviews in programming courses. Student feedback may not always be specific or match the evaluation criteria without a clear system. Using ML for automatic classification can provide automated feedback to reviewers, helping them align their feedback with the rubric before submission. This can encourage reviewers to give more precise and more thoughtful feedback. It can also make peer feedback more useful by ensuring authors receive feedback directly related to areas they can improve. In this way, the peer code review process can better support learning.

The results of the word cloud analysis provide valuable insights into student engagement and feedback strategies during their initial peer code review activities. Despite having limited prior experience, students demonstrated intentional effort and engaged in meaningful interactions with their peers' code submissions. Frequently mentioned terms such as "fungsi" (function), "pembagian" (division), "penamaan variabel" (variable naming), and "layout" suggest that students effectively focused on assessing critical dimensions of code quality, including documentation, modularity, readability, and functional correctness. This observation aligns with existing research emphasizing that structured peer review encourages students to critically engage with multiple aspects of programming assignments [77, 78].

However, the dominance of positive descriptors such as "baik" (good), "bagus" (great), and "rapi" (neat) suggests that students tended to provide more affirmations than detailed critiques, a pattern frequently observed in initial peer review exercises [78]. The relatively sparse use of critical terms like "kurang" (lacking) and "kurang optimal" (suboptimal) suggests students might have encountered difficulties articulating constructive, improvement-oriented feedback. Previous studies confirm this is a typical challenge during early-stage peer code reviews, stemming from students' discomfort or uncertainty in providing specific and actionable criticism [79]. To enhance students' peer code review capabilities, future interventions should focus on guiding students beyond surface-level affirmations toward detailed, actionable critiques. Effective strategies could include structured training sessions, balanced and constructive feedback exemplars, and explicit criteria for assessing code quality. Such structured support has been shown to improve the quality and specificity of peer feedback significantly, enabling students to articulate both strengths and areas for improvement clearly and effectively [8, 80].

After analyzing both confusion matrices for the best-performing ML (Random Forest) and DL (RCNN) models, specific areas of potential improvement emerge. The misclassifications observed in both models, particularly in labels such as "Flow Control", "Layout and Formatting", and "General" indicate challenges in capturing subtle syntactic and semantic distinctions between these closely related classes. To address these limitations, enhancing the feature representation might significantly improve classification accuracy. Prior research emphasizes that incorporating semantic features, such as contextual embeddings using BERT [81, 82] or IndoBERT [83], can better capture linguistic nuances that simple count-based features might overlook. Additionally, applying methods for handling class ambiguity, such as ensemble approaches or hybrid models combining the strengths of both ML and DL frameworks, has been shown to effectively reduce misclassification errors [84].

It is important to note the limitations related to external validity in this study. The dataset was collected from a single higher education institution. Student writing style, familiarity with the rubric, and communication patterns may differ across institutions and cultural contexts. Therefore, the findings should be interpreted with caution and not assumed to be universally generalizable. The primary goal of this study is to establish a methodological benchmark and provide a validated, rubric-annotated dataset in a low-resource language context, specifically Bahasa Indonesia. The proposed approach and experimental framework are intended to be transferable. They can be replicated or extended using data from multiple institutions. Future work may evaluate the models on multi-institutional and cross-cultural datasets to further examine robustness and generalizability.

This study develops and evaluates supervised ML, DL, and few-shot approaches for single-label classification of peer code review feedback. Despite these contributions, several limitations remain. The single-label setting aligns well with existing peer code review tools, which usually provide separate text boxes for each rubric criterion. In this setting, the model can indicate whether a student's feedback matches the intended rubric category and can support immediate revision.

However, student feedback may contain multiple ideas within a single comment. A comment written for one rubric criterion may also include content related to other criteria. In the current setting, the model can detect the presence of a relevant rubric concept. It cannot identify or separate different rubric-related segments within the same comment. This limits its ability to support a more fine-grained feedback organization.

Future work may explore sequence-based or span-level approaches, such as Named Entity Recognition, to identify and localize multiple rubric-related concepts within a single feedback instance. These approaches can provide more precise feedback guidance by linking specific parts of a comment to corresponding rubric criteria. They can also remain compatible with existing peer code review workflows.

From a technical perspective, future implementations can integrate the classification model into existing peer code

review platforms. The model can be deployed as a lightweight backend service. This service can process student feedback in real time and return the predicted rubric category before submission. For sequence-based extensions, Named Entity Recognition models can be used to identify rubric-related segments at the token or phrase level. The interface can then highlight relevant parts of the feedback. Feature-based models, such as Random Forest, can be combined with these components to maintain efficiency and interpretability. This design also supports deployment in low-resource educational settings.

Beyond rubric alignment, future studies could examine the qualitative dimensions of student feedback, such as whether students identify strengths or weaknesses, provide specific or general feedback, and offer actionable suggestions or vague praise. The word cloud analysis showed that students frequently use appreciative language (e.g., "baik", "sangat baik"), indicating a tendency toward positive tone. However, this study has not yet explored the sentiment or depth of feedback. Future research could apply sentiment and multi-label analysis to better distinguish between praise and critique, assess the specificity of feedback, and identify the presence of actionable advice; thereby deepening the system's understanding of peer code review quality and improving feedback training strategies.

D. Implication

This study makes two key theoretical contributions. It provides the novel labelled corpus of peer code review feedback in Bahasa Indonesia, annotated with rubric criteria to support cross-lingual research on educational feedback. The results also show that classical ML models, such as Random Forest with Count Vectorization, can outperform more complex DL and LLM approaches for short rubric-driven texts. This finding challenges the assumption that DL is always superior and highlights the continued value of simpler models in low-resource settings.

The study also offers practical implications for computing education. The Random Forest model runs efficiently on standard hardware, making it feasible for integration into peer code review tools that give students real-time feedback on rubric alignment. Analysis of word clouds and misclassified feedback further identified weaknesses in student feedback, such as reliance on vague praise rather than actionable critique. Educators can use these insights to design training activities that emphasize clarity and specificity.

A practical next step is to integrate the model into peer code review tools, providing students with real-time feedback on rubric alignment. Students can revise unclear or off-target feedback before submitting it. This support can help students produce comments that align more accurately with the rubric. It can also help instructors by improving the overall quality and consistency of peer feedback.

Beyond classification, the tool also supports learning. Peer review research shows that students improve evaluative judgment and feedback quality through practice and guided revision [85]. Research also shows that students benefit from clear guidance when learning to write specific and helpful comments [86]. The tool provides this guidance by signaling vague or off-target feedback in real time. Students can revise their comments immediately, which helps them learn to give more precise and focused feedback. Teachers can also use the system to show examples and support training activities. In this way, the tool not only classifies feedback but also helps students develop stronger feedback skills, increasing its pedagogical value and external validity.

V. Conclusion

This study demonstrates that automated classification can support more consistent peer code review feedback among students in introductory programming courses. By aligning feedback with rubric criteria, the approach helps address vague or unfocused comments and offers a basis for real-time, criterion-based guidance. The study contributes a corpus of 2281 peer code review feedback in Bahasa Indonesia and a systematic benchmark of ML, DL, and few-shot prompting models under identical conditions.

Random Forest with count vectors achieved the best performance, outperforming both DL and LLM-based approaches. This result challenges assumptions about the superiority of more advanced models and shows that classical ML remains effective for short, rubric-driven texts in low-resource settings. Misclassification patterns also reveal challenges in distinguishing closely related labels, such as Layout and Formatting and Flow Control, due to overlapping lexical cues. These findings help clarify how students express feedback and highlight areas where rubric distinctions may be less explicit in practice.

The analysis further shows that while students engage with core programming concepts, their feedback often relies on general praise rather than specific, actionable advice. This suggests the need for instructional support to enhance feedback practices in conjunction with the use of automated tools.

Future work may extend the proposed approach by using sequence-based analysis, such as Named Entity Recognition, to identify and localize multiple rubric-related concepts within a single feedback comment. Other directions include integrating sentiment analysis to capture feedback tone and specificity. Future studies may also evaluate real-time deployment within peer code review tools. These extensions can enhance rubric-based feedback systems and support more consistent and pedagogically meaningful peer code review processes. The results highlight the practical potential of automated classification to improve feedback quality and learning experiences in programming courses.

Appendix

PROMPT 1-SHOT

You are an assistant that classifies each peer–code-review feedback into **exactly one** of seven labels. After reading every feedback, write the predicted label in a new column named PredictedLabel. LABEL DEFINITIONS:

1. Variable: feedback specifically addressing whether variable names clearly reflect their intended purpose.

2. Expressions: feedback concerning the simplicity, appropriateness, and clarity of the expressions and data types used in the code.

3. Control Flow: feedback assessing the simplicity and appropriateness of control structures, including handling of exceptions and overall logic flow clarity.

4. Comments: feedback evaluating the clarity, completeness, spelling, and usefulness of header or inline comments within the code.

5. Layout and Formatting: feedback focused on readability—including code grouping, indentation, spacing, blank lines, and consistent use of brackets.

6. Decomposition: feedback related to how clearly and effectively the code is divided into routines/modules with distinct tasks, minimal duplication, and limited shared variables.

7. General: feedback that do not specifically align with any label above; general or overall feedback, or remarks about assignment specifications/code files.

OUTPUT RULES

* Input arrives as a CSV snippet that always starts with a header line: Feedback followed by one feedback per row.

* Return **the same CSV plus one extra column** called PredictedLabel

* Reply with CSV **only** – no additional explanation or formatting.

ONE-SHOT EXAMPLE

Feedback (Indonesian): "Penamaan variabel sudah cukup baik dan jelas. Setiap variabel bisa dibaca dengan jelas sehingga bisa mengetahui apa fungsi dari tiap variabel."

Label: Variable

DATA <<peer-code-review-dataset-test>>

#END OF PROMPT

CONFLICT OF INTEREST

The authors declare no conflict of interest.

AUTHOR CONTRIBUTIONS

Theresia Devi Indriasari: Conceptualization, Methodology, Software, Investigation, Data Curation, Writing - Original draft preparation, Writing - Review & Editing. Yohanes Sigit Purnomo W.P.: Conceptualization, Methodology, Software, Writing - Review & Editing. All authors had approved the final version.

ACKNOWLEDGMENT

The authors wish to thank Universitas Atma Jaya Yogyakarta and the University of Auckland for the support provided during the conduct of this study.

REFERENCES

[1] Y. Wang, G. J. Komol, and D. Voltz, "Peer critique in an online environment: Nature of student critiques and their perceptions of the process," *Journal of Educational Technology Systems*, vol. 52, no. 1, pp. 5–26, 2023. doi: 10.1177/00472395231175157

[2] C. Dominguez *et al.*, "Adding value to the learning process by online peer review activities: Towards the elaboration of a methodology to promote critical thinking in future engineers," *European Journal of Engineering Education*, vol. 40, no. 5, pp. 573–591, Sep. 2015. doi: 10.1080/03043797.2014.987649

[3] F. Janesarvatan and M. Asoodar, "Constructive peer-feedback to improve language and communication skills in medical education," *Innovation in Language Learning and Teaching*, 2024. doi: 10.1080/17501229.2024.2311834

[4] H.-C. Lin, G.-J. Hwang, S.-C. Chang, and Y.-D. Hsu, "Facilitating critical thinking in decision making-based professional training: An online interactive peer-review approach in a flipped learning context," *Comput Educ*, vol. 173, 104266, 2021. doi: 10.1016/j.compedu.2021.104266

[5] D. Reinholz, "The assessment cycle: a model for learning through peer assessment," *Assess Eval High Educ*, vol. 41, no. 2, pp. 301–315, 2016. doi: 10.1080/02602938.2015.1008982

[6] N. Ardill, "Peer feedback in higher education: Student perceptions of peer review and strategies for learning enhancement," *European Journal of Higher Education*, vol. 0, no. 0, pp. 1–26, 2025. doi: 10.1080/21568235.2025.2457466

[7] D. Nicol and S. McCallum, "Making internal feedback explicit: exploiting the multiple comparisons that occur during peer review," *Assess Eval High Educ*, vol. 47, no. 3, pp. 424–443, 2022. doi: 10.1080/02602938.2021.1924620

[8] T. D. Indriasari, A. Luxton-Reilly, and P. Denny, "A review of peer code review in higher education," *ACM Trans. Comput. Educ.*, vol. 20, no. 3, Sep. 2020. doi: 10.1145/3403935

[9] A. K. Turzo and A. Bosu, "What makes a code review useful to OpenDev developers? An empirical investigation," *Empir Softw Eng*, vol. 29, no. 1, Feb. 2024. doi: 10.1007/s10664-023-10411-x

[10] O. Kononenko, O. Baysal, and M. W. Godfrey, "Code review quality: How developers see it," in *Proc. International Conference on Software Engineering*, IEEE Computer Society, May 2016, pp. 1028–1038. doi: 10.1145/2884781.2884840

[11] C. Li, Z. Dong, R. H. Untch, and M. Chasteen, "Facilitating peer review in an online collaborative learning environment for computer science students," in *Proc. the International Conference on Frontiers in Education: Computer Science and Computer Engineering (FECS)*, Athens, 2012, pp. 1–7.

[12] K. Reily, P. L. Finnerty, and L. Terveen, "Two peers are better than one: aggregating peer reviews for computing assignments is surprisingly accurate," in *Proc. the 2009 ACM International Conference on Supporting Group Work*, in GROUP '09. New York, NY, USA: Association for Computing Machinery, 2009, pp. 115–124. doi: 10.1145/1531674.1531692

[13] M. H. Ismail, S. Ismail, M. N. A. Nor'a, J. J. Sijore, and A. H. A. Hamid, "A systematic literature review on recent peer code review implementation in education," in *Proc. 2024 International Conference on TVET Excellence & Development (ICTeD)*, 2024, pp. 13–19. doi: 10.1109/ICTeD62334.2024.10844661

[14] S. Strickroth, "Does peer code review change my mind on my submission?" in *Proc. Annual Conference on Innovation and Technology in Computer Science Education, ITiCSE*, Association for Computing Machinery, Jun. 2023, pp. 498–504. doi: 10.1145/3587102.3588802

[15] M. M. Patchan, C. D. Schunn, and R. J. Correnti, "The nature of feedback: How peer feedback features affect students' implementation rate and quality of revisions," *J Educ Psychol*, vol. 108, no. 8, pp. 1098–1120, Nov. 2016. doi: 10.1037/edu0000103

[16] A. Dood, K. Das, Z. Qian, S. Finkenstaedt-Quinn, A. Gere, and G. Shultz, "A dashboard to provide instructors with automated feedback on students' peer review comments," in *ACM International Conference Proceeding Series*, Association for Computing Machinery, Mar. 2023, pp. 619–625. doi: 10.1145/3576050.3576087

[17] W. Hart-Davidson, R. Omizo, and M. Meeks, "Detecting high-quality comments in written feedback with a zero shot classifier," in *Proc. the 39th ACM International Conference on the Design of Communication: Building Coalitions. Worldwide, SIGDOC 2021*, Association for Computing Machinery, Inc, Oct. 2021, pp. 319–325. doi: 10.1145/3472714.3473659

[18] M. P. Ortega, L. B. Mendoza, J. M. Hormaza, and S. V. Soto, "Accuracy' measures of sentiment analysis algorithms for Spanish corpus generated in peer assessment," in *Proc. the 6th International Conference on Engineering & MIS 2020*, Association for Computing Machinery, Sep. 2020, pp. 1–7. doi: 10.1145/3410352.3410838

[19] Y. Xiao *et al.*, "Modeling review helpfulness with augmented transformer neural networks," in *Proc. 16th IEEE International Conference on Semantic Computing, ICSC 2022*, Institute of Electrical and Electronics Engineers Inc., 2022, pp. 83–90. doi: 10.1109/ICSC52841.2022.00019

[20] X. Liu and L. Li, "Assessment training effects on student assessment skills and task performance in a technology-facilitated peer assessment," *Assess Eval High Educ*, vol. 39, no. 3, pp. 275–292, 2014. doi: 10.1080/02602938.2013.823540

[21] M. Fernández-Toro and C. Furnborough, "Evaluating alignment of student and tutor perspectives on feedback on language learning assignments," *Distance Education*, vol. 39, no. 4, pp. 548–567, 2018. doi: 10.1080/01587919.2018.1520043

[22] M. M. Patchan, C. D. Schunn, and R. J. Clark, "Accountability in peer assessment: examining the effects of reviewing grades on peer ratings and peer feedback," *Studies in Higher Education*, vol. 43, no. 12, pp. 2263–2278, 2018. doi: 10.1080/03075079.2017.1320374

[23] M. P. Rashid, E. Gehringer, and H. Khosravi, "Navigating (dis)agreement: AI assistance to uncover peer feedback discrepancies," in *Proc. the 14th Learning Analytics and Knowledge Conference*, in

LAK '24. New York, NY, USA: Association for Computing Machinery, 2024, pp. 907–914. doi: 10.1145/3636555.3636931

[24] D. Babik, E. Gehringer, J. Kidd, K. Sunday, D. Tinapple, and S. Gilbert, "A systematic review of educational online peer-review and assessment systems: charting the landscape," *Educational Technology Research and Development*, vol. 72, no. 3, pp. 1653–1689, 2024. doi: 10.1007/s11423-024-10349-x

[25] C. J. Huang, Y. W. Wang, S. C. Chang, S. Y. Lin, J. H. Tseng, and J. J. Jian, "Applications of data mining to an online argumentation based learning assistance platform," in *Proc. 6th International Conference on Soft Computing and Intelligent Systems, and 13th International Symposium on Advanced Intelligence Systems, SCIS/ISIS 2012*, 2012, pp. 807–811. doi: 10.1109/SCIS-ISIS.2012.6505083

[26] E. Scheihing, M. Vernier, J. Guerra, J. Born, and L. Cárcamo, "Understanding the role of micro-blogging in b-learning activities: Kelluwen experiences in Chilean public schools," *IEEE Transactions on Learning Technologies*, vol. 11, no. 3, pp. 280–293, Jul. 2018. doi: 10.1109/TLT.2017.2714163

[27] Y. Xiao, Y. Gao, C. Yue, and E. Gehringer, "Estimating student grades through peer assessment as a crowdsourcing calibration problem," in *Proc. 2022 20th International Conference on Information Technology Based Higher Education and Training, ITHET 2022*, Antalya, Turkey: Institute of Electrical and Electronics Engineers Inc., 2022, pp. 1–9. doi: 10.1109/ITHET56107.2022.10031993

[28] M. P. Rashid, E. F. Gehringer, M. Young, D. Doshi, Q. Jia, and Y. Xiao, "Peer assessment rubric analyzer: An NLP approach to analyzing rubric items for better," in *Proc. 2021 19th International Conference on Information Technology Based Higher Education and Training, ITHET 2021*, Sydney, Australia: IEEE, Nov. 2021. doi: 10.1109/ITHET50392.2021.9759679

[29] S. Berrezueta–Guzman, S. Krusche, and S. Wagner, "From coders to critics: Empowering students through peer assessment in the age of AI copilots," in *Proc. 2025 International Symposium on Educational Technology (ISET)*, 2025, pp. 66–71.

[30] A. Pathak *et al.*, "Rubric is all you need: Improving LLM-based code evaluation with question-specific rubrics," in *Proc. the 2025 ACM Conference on International Computing Education Research V.1*, in ICER '25. New York, NY, USA: Association for Computing Machinery, 2025, pp. 181–195. doi: 10.1145/3702652.3744220

[31] Z. Li *et al.*, "Automating code review activities by large-scale pre-training," in *Proc. the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, in ESEC/FSE 2022. New York, NY, USA: Association for Computing Machinery, 2022, pp. 1035–1047. doi: 10.1145/3540250.3549081

[32] M. Stegeman, E. Barendsen, and S. Smetsers, "Designing a rubric for feedback on code quality in programming courses," in *ACM International Conference Proceeding Series*, pp. 160–164, 2016. doi: 10.1145/2999541.2999555

[33] R. Meléndez, M. Ptaszynski, and F. Masui, "Comparative investigation of traditional machine-learning models and transformer models for phishing email detection," *Electronics (Switzerland)*, vol. 13, no. 24, Dec. 2024. doi: 10.3390/electronics13244877

[34] S. Minaee, N. Kalchbrenner, E. Cambria, N. Nikzad, M. Chenaghlu, and J. Gao, "Deep learning-based text classification: A comprehensive review," *ACM Comput Surv*, vol. 54, no. 3, pp. 1–40, Apr. 2021. doi: 10.1145/3439726

[35] P. Liu, W. Yuan, J. Fu, Z. Jiang, H. Hayashi, and G. Neubig, "Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing," *ACM Comput. Surv.*, vol. 55, no. 9, Jan. 2023. doi: 10.1145/3560815

[36] K. Guo, E. D. Zhang, D. Li, and S. Yu, "Using AI-supported peer review to enhance feedback literacy: An investigation of students' revision of feedback on peers' essays," *British Journal of Educational Technology*, vol. 56, no. 4, pp. 1612–1639, 2025. doi: https://doi.org/10.1111/bjet.13540

[37] C. Liu, J. Cui, R. Shang, Y. Xiao, Q. Jia, and E. Gehringer, "Improving problem detection in peer assessment through pseudo-labeling using semi-supervised learning," *International Educational Data Mining Society*, 2022.

[38] A. Dood, B. Winograd, S. Finkenstaedt-Quinn, A. Gere, and G. Shultz, "PeerBERT: Automated characterization of peer review comments across courses," in *Proc. LAK22: 12th International Learning Analytics and Knowledge Conference*, in LAK22. New York, NY, USA: Association for Computing Machinery, 2022, pp. 492–499. doi: 10.1145/3506860.3506892

[39] G. and M. C. Amendola Daniela and Nalli, "Machine-learning-based software to group heterogeneous students for online peer assessment activities," in *Higher Education Learning Methodologies and Technologies Online*, D. and C. G. and C. M. and L. B. G. and T. D. Fulantelli Giovanni and Burgos, Ed., Cham: Springer Nature Switzerland, 2023, pp. 17–29.

[40] S. Basuki, Z. Sari, M. Tsuchiya, and R. Indrabayu, "Predicting the sentiment of review aspects in the peer review text using machine learning," *Kinetik: Game Technology, Information System, Computer Network, Computing, Electronics, and Control*, vol. 9, no. 4, Nov. 2024. doi: 10.22219/kinetik.v9i4.2042

[41] A. Koufakou, "Deep learning for opinion mining and topic classification of course reviews," *Educ Inf Technol (Dordr)*, vol. 29, no. 3, pp. 2973–2997, 2024. doi: 10.1007/s10639-023-11736-2

[42] G. Okasa *et al.*, "A supervised machine learning approach for assessing grant peer review reports," *Quantitative Science Studies*, pp. 1–40, Sep. 2025. doi: 10.1162/qss.a.23

[43] Z. Ersozlu, S. Taheri, and I. Koch, "A review of machine learning methods used for educational data," *Educ Inf Technol (Dordr)*, vol. 29, no. 16, pp. 22125–22145, 2024. doi: 10.1007/s10639-024-12704-0

[44] K. Huang, R. Ferreira Mello, C. Pereira Junior, L. Rodrigues, M. Baars, and O. Viberg, "That's what RoBERTa said: Explainable classification of peer feedback," in *Proc. the 15th International Learning Analytics and Knowledge Conference*, in LAK '25. New York, NY, USA: Association for Computing Machinery, 2025, pp. 880–886. doi: 10.1145/3706468.3706526

[45] D. A. Joyner, G. J. Pérez Irizarry, B. L. Eicher, and N. Lytle, "Intelligent feedback and evaluation for the assessment and improvement of student peer reviews," in *Proc. 2024 IEEE Digital Education and MOOCS Conference (DEMOcon)*, 2024, pp. 1–6. doi: 10.1109/DEMOcon63027.2024.10747900

[46] T. M. Adeyanju, A. A. Aziz, and S. Safei, "A comparative analysis of classification algorithms on student academic performance," *Semarak International Journal of Machine Learning*, vol. 5, no. 1, pp. 74–86, 2025.

[47] M. Bin Roslan and C. Chen, "Educational data mining for student performance prediction: A systematic literature review (2015–2021)," *International Journal of Emerging Technologies in Learning (iJET)*, vol. 17, no. 5, pp. 147–179, 2022.

[48] N. A. Butt, Z. Mahmood, K. Shakeel, S. Alfarhood, M. Safran, and I. Ashraf, "Performance prediction of students in higher education using multi-model ensemble approach," *IEEE Access*, vol. 11, pp. 136091–136108, 2023. doi: 10.1109/ACCESS.2023.3336987

[49] H. Y. Phuong, Q. T. Phan, and T. T. Le, "The effects of using analytical rubrics in peer and self-assessment on EFL students' writing proficiency: a Vietnamese contextual study," *Language Testing in Asia*, vol. 13, no. 1, p. 42, 2023. doi: 10.1186/s40468-023-00256-y

[50] S. Lertsakulbunlue and A. Kantiwong, "Development of peer assessment rubrics in simulation-based learning for advanced cardiac life support skills among medical students," *Advances in Simulation*, vol. 9, no. 1, p. 25, 2024. doi: 10.1186/s41077-024-00301-7

[51] S. J. Kruger, "The reliability of peer assessment in a final–year information systems course," *South African Journal of Higher Education*, vol. 39, no. 1, pp. 188–205, 2025.

[52] S. Thite, J. Ravishankar, I. Tomeo-Reyes, and A. M. Ortiz, "Design of a simple rubric to peer-evaluate the teamwork skills of engineering students," *European Journal of Engineering Education*, vol. 49, no. 4, pp. 623–646, 2024. doi: 10.1080/03043797.2024.2338239

[53] S. M. Fadillah and M. Ha, "Using rubrics to enhance accuracy of peer assessment and self-judgment in biology learning," *Asia-Pacific Science Education*, vol. 10, no. 2, pp. 265–288, 2024. doi: 10.1163/23641177-bja10080

[54] L. Mphahlele, "Students' perception of the use of a rubric and peer reviews in an online learning environment," *Journal of Risk and Financial Management*, vol. 15, no. 11, 2022. doi: 10.3390/jrfm15110503

[55] R. Bacchus and J. Wallace, "Peer assessment using student co-designed rubrics," *Creat Educ*, vol. 15, no. 2, pp. 164–177, 2024.

[56] H. Gong and others, "The impact of co-creating rubrics on peer assessment in higher education EFL classes," *International Journal of New Developments in Education*, vol. 5, no. 23, pp. 153–161, 2023.

[57] W. Handayani, H. Saptopramono, and D. Apriyanti, "Lecturers' perceptions on the utilization of translation rubrics for guiding peer and self-assessment in higher education," *Indonesian Journal of Integrated English Language Teaching*, vol. 11, no. 1, pp. 1–9, 2025.

[58] T. D. Indriasari, P. Denny, D. Lottridge, and L.-R. Andrew, "Gamification improves the quality of student peer code review," *Computer Science Education*, vol. 33, no. 3, pp. 458–482, 2023. doi: 10.1080/08993408.2022.2124094

[59] S. Strickroth and I. Azaiz, "Qualitative analysis of peer reviews of a large introductory programming course," *Computer Science Education*, vol. 0, no. 0, pp. 1–21, 2025. doi: 10.1080/08993408.2025.2450587

[60] S. Esche, "Rubric for the quality of answers to student queries about code," in *Proc. the 55th ACM Technical Symposium on Computer*

Science Education V. 1, in SIGCSE 2024, New York, NY, USA: Association for Computing Machinery, 2024, pp. 331–337. doi: 10.1145/3626252.3630918

[61] M. Stegeman, E. Barendsen, and S. Smetsers, "Towards an empirically validated model for assessment of code quality," *ACM International Conference Proceeding Series*, vol. 2014-Novem, no. November, pp. 99–108, 2014. doi: 10.1145/2674683.2674702

[62] A. Amalia, O. S. Sitompul, E. B. Nababan, and T. Mantoro, "An efficient text classification using fastText for Bahasa Indonesia documents classification," in *Proc. 2020 International Conference on Data Science, Artificial Intelligence, and Business Analytics (DATABIA)*, 2020, pp. 69–75. doi: 10.1109/DATABIA50434.2020.9190447

[63] R. Artstein and M. Poesio, "Inter-coder agreement for computational linguistics," *Computational Linguistics*, vol. 34, no. 4, pp. 555–596, 2008. doi: 10.1162/coli.07-034-R2

[64] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov, "Enriching word vectors with subword information," *Trans Assoc Comput Linguist*, vol. 5, pp. 135–146, 2017. doi: 10.1162/tacl_a_00051

[65] O. Rainio, J. Teuho, and R. Klén, "Evaluation metrics and statistical tests for machine learning," *Sci Rep*, vol. 14, no. 1, Dec. 2024. doi: 10.1038/s41598-024-56706-x

[66] H. Chen, L. Wu, J. Chen, W. Lu, and J. Ding, "A comparative study of automated legal text classification using random forests and deep learning," *Inf Process Manag*, vol. 59, no. 2, 102798, 2022. doi: 10.1016/j.ipm.2021.102798

[67] Q. Li *et al.*, "A survey on text classification: From traditional to deep learning," *ACM Trans Intell Syst Technol*, vol. 13, no. 2, pp. 1–41, Apr. 2022. doi: 10.1145/3495162

[68] M. T. Ribeiro, S. Singh, and C. Guestrin, "'Why should I trust you?': Explaining the predictions of any classifier," in *Proc. the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, in KDD '16. New York, NY, USA: Association for Computing Machinery, 2016, pp. 1135–1144. doi: 10.1145/2939672.2939778

[69] S. M. Lundberg and S.-I. Lee, "A unified approach to interpreting model predictions," in *Proc. the 31st International Conference on Neural Information Processing Systems*, in NIPS'17. Red Hook, NY, USA: Curran Associates Inc., 2017, pp. 4768–4777.

[70] N. T. Kerman, S. K. Banihashem, M. Karami, E. Er, S. van Ginkel, and O. Noroozi, "Online peer feedback in higher education: A synthesis of the literature," *Educ Inf Technol (Dordr)*, vol. 29, no. 1, pp. 763–813, Jan. 2024. doi: 10.1007/s10639-023-12273-8

[71] X. Jin, Q. Jiang, W. Xiong, Y. Feng, and W. Zhao, "Effects of student engagement in peer feedback on writing performance in higher education," *Interactive Learning Environments*, vol. 32, no. 1, pp. 128–143, 2024. doi: 10.1080/10494820.2022.2081209

[72] K. Guo, M. Pan, Y. Li, and C. Lai, "Effects of an AI-supported approach to peer feedback on university EFL students' feedback quality and writing ability," *Internet High Educ*, vol. 63, 100962, 2024. doi: 10.1016/j.iheduc.2024.100962

[73] A. Valero Haro, O. Noroozi, H. J. A. Biemans, M. Mulder, and S. K. Banihashem, "How does the type of online peer feedback influence feedback quality, argumentative essay writing quality, and domain-specific learning?" *Interactive Learning Environments*, vol. 32, no. 9, pp. 5459–5478, 2023. doi: 10.1080/10494820.2023.2215822

[74] X. Gao, O. Noroozi, J. Gulikers, H. J. A. Biemans, and S. K. Banihashem, "A systematic review of the key components of online peer feedback practices in higher education," *Educational Research Review*, 2024. doi: 10.1016/j.edurev.2023.100588

[75] S. Latifi, O. Noroozi, and E. Talaee, "Peer feedback or peer feedforward? Enhancing students' argumentative peer learning processes and outcomes," *British Journal of Educational Technology*, vol. 52, no. 2, pp. 768–784, Mar. 2021. doi: 10.1111/bjet.13054

[76] O. Noroozi, S. K. Banihashem, H. J. A. Biemans, M. Smits, M. T. W. Vervoort, and C. L. Verbaan, "Design, implementation, and evaluation of an online supported peer feedback module to enhance students' argumentative essay quality," *Educ Inf Technol (Dordr)*, vol. 28, no. 10, pp. 12757–12784, Oct. 2023. doi: 10.1007/s10639-023-11683-y

[77] T. Brown, G. S. Walia, A. D. Radermacher, M. Singh, and M. R. Narasareddygari, "Effectiveness of using guided peer code review to support learning of programming concepts in a CS2 course: A pilot study," in *Proc. 2020 ASEE Virtual Annual Conference Content Access*, Jun. 2020. doi: 10.18260/1-2--34504

[78] R. Sabarinath and C. L. G. Quek, "A case study investigating programming students' peer review of codes and their perceptions of the online learning environment," *Educ Inf Technol (Dordr)*, vol. 25, no. 5, pp. 3553–3575, 2020. doi: 10.1007/s10639-020-10111-9

[79] K. Fleischmann, "Making tacit knowledge explicit: the case for online peer feedback in the studio critique," *Int J Technol Des Educ*, vol. 35, no. 2, pp. 699–721, Apr. 2025. doi: 10.1007/s10798-024-09911-8

[80] C. Y. Chong, P. Thongtanunam, and C. Tantithamthavorn, "Assessing the students' understanding and their mistakes in code review checklists: an experience report of 1,791 code review checklist questions from 394 students," in *Proc. the 43rd International Conference on Software Engineering: Joint Track on Software Engineering Education and Training*, in ICSE-JSEET '21. IEEE Press, 2021, pp. 20–29. doi: 10.1109/ICSE-SEET52601.2021.00011

[81] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in *Proc. the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, J. Burstein, C. Doran, and T. Solorio, Eds., Minneapolis, Minnesota: Association for Computational Linguistics, Jun. 2019, pp. 4171–4186. doi: 10.18653/v1/N19-1423

[82] T. Mikolov, K. Chen, G. S. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," in Proc. *International Conference on Learning Representations*, Jan. 2013.

[83] B. Wilie *et al.*, "IndoNLU: Benchmark and resources for evaluating Indonesian natural language understanding," in *Proc. the 1st Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics and the 10th International Joint Conference on Natural Language Processing*, K.-F. Wong, K. Knight, and H. Wu, Eds., Suzhou, China: Association for Computational Linguistics, Dec. 2020, pp. 843–857. doi: 10.18653/v1/2020.aacl-main.85

[84] O. Sagi and L. Rokach, "Ensemble learning: A survey," *WIREs Data Mining and Knowledge Discovery*, vol. 8, no. 4, e1249, 2018. doi: 10.1002/widm.1249

[85] D. J. Nicol and D. Macfarlane-Dick, "Formative assessment and self-regulated learning: A model and seven principles of good feedback practice," *Studies in Higher Education*, vol. 31, no. 2, pp. 199–218, 2006. doi: 10.1080/03075070600572090

[86] D. Carless and D. Boud, "The development of student feedback literacy: enabling uptake of feedback," *Assess Eval High Educ*, vol. 43, no. 8, pp. 1315–1325, 2018. doi: 10.1080/02602938.2018.1463354

Copyright © 2026 by the authors. This is an open access article distributed under the Creative Commons Attribution License which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited (CC BY 4.0).