future internet

MDPI

Article

JorGPT: Instructor-Aided Grading of Programming Assignments with Large Language Models (LLMs)

Jorge Cisneros-González, Natalia Gordo-Herrera, Iván Barcia-Santos and Javier Sánchez-Soriano *

Advanced Artificial Intelligence Group (A^{2}IG), Escuela Politécnica Superior, Universidad Francisco de Vitoria, 28223 Pozuelo de Alarcón, Madrid, Spain; j.cisneros.prof@ufv.es (J.C.-G.); natalia.gordo@ufv.es (N.G.-H.); i.barcia@ufv.es (I.B.-S.)

* Correspondence: javier.sanchez@ufv.es

Abstract: This paper explores the application of large language models (LLMs) to automate the evaluation of programming assignments in an undergraduate "Introduction to Programming" course. This study addresses the challenges of manual grading, including time constraints and potential inconsistencies, by proposing a system that integrates several LLMs to streamline the assessment process. The system utilizes a graphic interface to process student submissions, allowing instructors to select an LLM and customize the grading rubric. A comparative analysis, using LLMs from OpenAI, Google, DeepSeek and ALIBABA to evaluate student code submissions, revealed a strong correlation between LLM-generated grades and those assigned by human instructors. Specifically, the reduced model using statistically significant variables demonstrates a high explanatory power, with an adjusted R^2 of 0.9156 and a Mean Absolute Error of 0.4579, indicating that LLMs can effectively replicate human grading. The findings suggest that LLMs can automate grading when paired with human oversight, drastically reducing the instructor workload, transforming a task estimated to take more than 300 h of manual work into less than 15 min of automated processing and improving the efficiency and consistency of assessment in computer science education.

Keywords: academic assessment; automated assessment; generative artificial intelligence; large language models; automated code assessment; AI-helped feedback; code grading

Academic Editors: Athanasios D. Panagopoulos and Filipe Portela

Received: 30 May 2025
Revised: 12 June 2025
Accepted: 16 June 2025
Published: 18 June 2025

Citation: Cisneros-González, J.; Gordo-Herrera, N.; Barcia-Santos, I.; Sánchez-Soriano, J. JorGPT: Instructor-Aided Grading of Programming Assignments with Large Language Models (LLMs). *Future Internet* **2025**, *17*, 265. https://doi.org/10.3390/fi17060265

1. Introduction

Assessment is a fundamental component of academic training. In recent years, this process has been the focus of debate, partly due to a perceived decline in the authority of teaching staff [1]. One of the strategies adopted within academia to address this has been the use of multiple-choice tests as an assessment model [2]. However, such tests have evident shortcomings in disciplines that require students to solve complex tasks, such as programming. To better handle this complexity and promote fairer, more consistent evaluation, assessment rubrics have been introduced. These tools make it possible to define various levels of achievement for each evaluation criterion [3], thereby aiming to reduce bias and subjectivity by structuring the assessment process. Yet, in courses such as "Introduction to Programming", two further challenges emerge.

In subjects of this nature, students' competencies improve significantly when they complete many graded assignments [4]. In contrast, ungraded exercises tend to be undertaken only by the highest-performing students and, generally, do not contribute to a significant improvement in overall competency.

Each graded programming assignment entails a considerable workload for instructors. It is estimated that manually assessing a programming task in introductory university courses takes on average 30 min per student [4].

In a standard semester-long course with 11 effective weeks for graded assignments and a group of 120 students, this results in roughly 60 h of grading per assignment each week, to be distributed among the teaching staff. Such a workload not only limits the number of assignments that can be realistically implemented, but also introduces the risk of inconsistent evaluation, even when rubrics are applied, due to fatigue or time constraints [5]. While increasing the number of instructors responsible for grading might appear to be a potential solution, this option proves unviable from both logistical and economic standpoints. Consequently, it becomes essential to explore automated assessment approaches that uphold the necessary pedagogical standards.

Recent applications of LLMs demonstrate their versatility across diverse domains. For instance, TrumorGPT applies LLMs and graph-based retrieval to fact-check health-related claims in real time [6], while ChatDL enhances software defect localization in IIoT manufacturing by combining LLMs with information retrieval techniques [7]. These examples highlight the growing impact of LLMs beyond general-purpose NLP, supporting their integration in domain-specific tasks. Our work follows this trend by applying LLMs to the field of computer science education, focusing on automating and improving the grading process for programming assignments. From a technical perspective, the partial automation of the assessment process through Generative Artificial Intelligence (GenAI) technologies presents itself as a viable and promising alternative [8]. A typical use of GenAI lies in its capacity to identify and correct errors in source code across a range of programming languages [9]. This includes detecting syntax mistakes, logical errors, or inefficient coding patterns, as well as offering suggestions to enhance the quality and efficiency of the code. Nevertheless, a valid concern in this context is the potential for algorithmic bias. This must be weighed alongside the reality that human grading is also susceptible to variability [10].

Although rubrics help to standardize evaluation, factors such as grader fatigue, high submission volume, or difference of criteria among instructors can still affect the overall objectivity of the process. Teaching experience indicates that, even when detailed rubrics are in place, there is often a tendency to assign grades using fixed intervals (e.g., scores from 0 to 5 in predefined blocks), which may not always capture the true quality of the submitted work. One way to address these issues is to incorporate the rubric directly as part of the prompt when working with generative models, allowing the system to adjust its responses according to the defined instructional guidelines [2]. In addition, a subsequent review by the teaching staff can act as a supervisory mechanism to ensure consistency and reliability in the system's output.

The main objective of this study is to examine the use of various large language models (LLMs) for automated evaluation of programming tasks. Our analysis includes models developed by organizations such as OpenAI (GPT 4o, GPT 4.1, GPT-o3-mini), Google (Gemini 2.0 Flash), DeepSeek, and ALIBABA (QWEN-plus), which were applied to evaluate programming exercises of an undergraduate "Introduction to Programming" course. More specifically, this paper introduces the system used and presents preliminary results of a comparative study of these models, highlighting initial challenges encountered and outlining possible directions for future development and refinement.

The remainder of this article is structured as follows. Section 2 reviews the existing literature, covering conventional approaches to code assessment, previous studies on the integration of Artificial Intelligence into academic evaluation, and the current limitations and challenges faced by automated grading systems. Section 3 offers an in-depth description of the implemented system, detailing its architecture—specifically designed for compatibility

with virtual learning environments—and explaining the workflow that enables instructors to export, submit, and process assignments for evaluation. Section 4 presents the results of our study, including a quantitative comparison between the grades assigned by eight different AI models and those given by human instructors for a dataset comprising 672 students. It also includes a qualitative analysis of the feedback generated by the models across submissions of varying quality. Section 5 focuses on the interpretation and discussion of the findings. Finally, Section 6 concludes the paper by summarizing the main contributions and proposing directions for future research, grounded in the insights obtained throughout this study.

2. Related Work

Prior to the emergence of AI-based grading, programming assignments were typically evaluated using manual or semi-automated methods, including the use of test cases, trace analysis, rubrics, and unit testing frameworks such as JUnit or PyTest to verify functional correctness [11–15]. Although these methods have proven effective, they are time-intensive and demand substantial manual effort from instructors. The following summary outlines the main characteristics of four commonly employed assessment methods.

Various strategies have been employed for code evaluation in educational settings, each with distinct advantages, limitations, and appropriate use cases [16]. One of the most prominent approaches is the use of test cases, valued for being objective, automatable, and capable of rapidly assessing the functional behavior of code [17]. Although well-designed test cases can infer the existence of logical errors, they often cannot directly evaluate the quality of internal reasoning or code style. Consequently, it is best suited for assignments where functional correctness is the primary focus. The feedback generated by this method is typically binary (pass/fail), although it can be enhanced through advanced tools and customized error messages designed to guide students [18].

This technique is especially valuable for assessing a student's understanding and for identifying flaws in control-flow logic. It offers detailed feedback on logical decision-making, flow-related errors, and the student's grasp of program structure [16]. Trace analysis is particularly effective for small-scale exercises or exam settings involving pseudocode or basic programs. However, it is inherently subjective and time-consuming, which limits its scalability in large groups.

Rubrics offer a flexible and detailed assessment framework that also promotes good programming practices. This approach is well suited for larger projects or final evaluations, as it enables a qualitative analysis of multiple dimensions of the code. Nevertheless, its implementation can be time-consuming and is sensitive to variations in evaluator consistency [17]. The feedback generated through rubrics typically highlights strengths and weaknesses, along with specific suggestions linked to each assessment criterion.

Finally, unit testing is distinguished by its precision and reusability, and it encourages the adoption of test-driven development practices. While it requires initial setup and ongoing maintenance and entails a learning curve, it is particularly appropriate for advanced courses or contexts focused on professional software testing methodologies. The feedback it provides is technical in nature, reporting function-specific errors and indicating the success or failure of each individual test.

2.1. AI in Academic Assessment

The application of Artificial Intelligence (AI) to academic assessment has gained growing interest in recent years. Traditional evaluation methods depend heavily on human graders, making them time-consuming and susceptible to subjectivity. AI-driven approaches seek to automate and improve assessment processes, aiming to enhance both

efficiency and consistency. This section reviews previous research on AI-based evaluation systems, with particular emphasis on machine learning (ML) techniques and large language models (LLMs).

Early AI-based assessment systems were designed for objective grading tasks, such as scoring multiple-choice questions and detecting plagiarism [19]. Over time, these approaches were extended to more complex domains, including automated essay evaluation [20] and the assessment of programming assignments [20]. The incorporation of deep learning techniques has significantly enhanced performance in these areas, allowing for improved understanding of both textual and structured inputs.

Machine learning techniques have been widely applied to the evaluation of student submissions, particularly in areas such as Natural Language Processing (NLP) and code analysis. Traditional models such as Support Vector Machines (SVMs) and Random Forests have been used for automated essay grading [21], while deep learning architectures—such as Recurrent Neural Networks (RNNs) and Transformers—have demonstrated superior performance in interpreting complex responses [22]. In the context of programming assessment, methods including code similarity analysis [23] and execution-based grading [24] have been proposed to enhance the effectiveness of student solution evaluation.

Recent advances in large language models (LLMs), such as OpenAI's GPT-4 [25] and Google's PaLM [26], have opened new avenues for automated assessment. These models have been utilized to deliver student feedback [27] and to evaluate open-ended responses [28]. While LLMs provide enhanced contextual understanding, challenges persist in ensuring fairness, reducing bias, and maintaining pedagogical rigor [29].

A recent contribution to this area is AI-PAT [30], an LLM-based system designed for exam evaluation and the management of student appeals within computer science education. An analysis of over 850 assessments and 185 appeals revealed correlations between models such as ChatGPT and Gemini, although prompt design significantly influenced result variability. The study uncovered disagreements among human graders and a high rate of grade changes (74%) after appeals, underscoring the subjectivity inherent in manual grading and the necessity to enhance automated systems. While students valued the speed and detail of AI-generated feedback, concerns regarding trust and fairness were also expressed. The authors highlight that AI-PAT is scalable but requires human oversight, well-defined rubrics, and effective appeal procedures to ensure fair outcomes.

2.2. Limitations and Challenges in Automated Assessment

Despite advances in AI-driven assessment, several challenges persist. A major concern is bias in automated grading, where AI models may preferentially favor certain writing styles or coding patterns [31]. Another limitation is the lack of interpretability, as deep learning models often operate as black boxes, complicating educators' understanding of grading decisions [32]. Ethical issues related to data privacy and the influence of automated grading on students' learning experiences also arise [33]. Building on existing AI assessment methods, our research compares multiple models to improve grading accuracy and feedback quality. Unlike prior work focusing on single evaluation techniques, our approach integrates several specialized LLMs to enhance performance across diverse task types. This study seeks to overcome current challenges by systematically comparing the behavior and biases of different black box models in grading and delivering personalized feedback to students.

Faced with this landscape, a significant gap emerges: the absence of unified frameworks that not only integrate but also systematically compare the performance of various state-of-the-art LLMs under identical conditions for educational assessment. While many studies focus on the application of a single model, our work addresses this gap by proposing

a system that (1) allows for a direct comparative analysis of multiple LLMs from different providers, (2) seamlessly integrates into the typical workflow of an instructor using a Learning Management System (LMS) such as Canvas, and (3) introduces a regression-based calibration model to combine AI scores, aiming for a final grade that is more robust and aligned with human expert judgment than the output of any single model.

3. Materials and Methods

3.1. General Approach

To address the challenges of consistency, transparency, and feedback quality outlined in the previous section, the primary objective of this research was to design and validate an automated evaluation system for programming assignments in an undergraduate "Introduction to Programming" course, leveraging large language models (LLMs). The primary goal was to replicate the assessment standards typically applied by human instructors, thereby reducing manual grading workload without compromising evaluation quality. The system was developed to integrate seamlessly with the Canvas learning management system, facilitating a more efficient grading pipeline.

An overview of the system's operation is illustrated in Figure 1. The assessment workflow begins with the export of student submissions from Canvas in CSV format. This dataset is processed through a graphical user interface built with PyQT5, where instructors can select the desired LLM for evaluation and optionally edit the grading rubric. Once configured, the system automatically analyzes each submission using the selected model, generating both grades and formative feedback. The results are stored in an Excel file for instructor review and, if needed, possible revision before final feedback is delivered to students via Canvas, as shown in Figure 1. This approach enables scalable and consistent evaluation across large groups of students, while preserving instructor oversight.

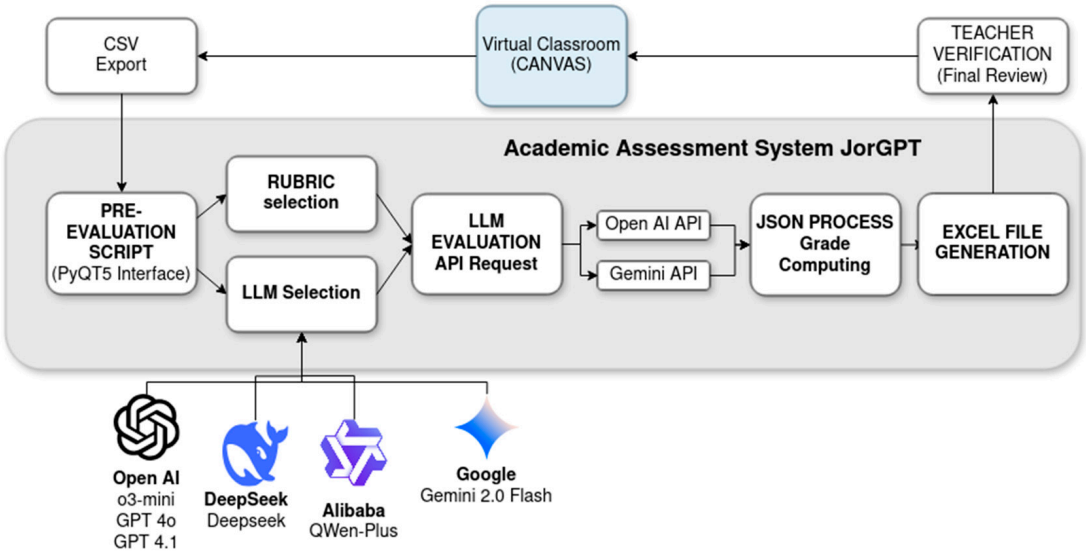

Figure 1. Data flows between the different system components and the LLMs.

3.2. Dataset and Data Preprocessing

The programming exercises used in this study came from an undergraduate "Introduction to Programming" course; these were real student assignments. The dataset comprised 672 student submissions. Of these, 80% were allocated for model training using cross-validation, and the remaining 20% were reserved for validating the proposed approach. The exercises are designed to assess fundamental programming concepts. The specific types of exercises include loops, conditional statements, functions, and basic data structures,

among others. The programming language used for the exercises was C, but it could be any other language, such as C++, Java, or Python. The student submissions were evaluated and graded by the course instructors, and these grades serve as the reference for evaluating the LLM-based assessment system.

The student submissions were exported from the Canvas virtual classroom as a CSV file. The system processes the data and generates results in JSON format and an Excel file.

The student code is read from a CSV file. The names of the comments are removed from the source code, while in the CSV they are used for internal management but are not passed on to the LLM to ensure anonymity. Exercises that are blank are skipped to avoid unnecessary API requests. To minimize the risk of prompt injection [28], the system adds <CODE></CODE> tags around the code.

Prior to submission to the LLM, the students' code undergoes minimal preprocessing. No tokenization or whitespace normalization is applied, as the goal is to preserve the original formatting for pedagogical evaluation. The programs are represented as plain source code.

3.3. LLM Assessment Systems

This study used several LLMs from different providers: OpenAI (GPT 4o, GPT 4.1, o3-mini), Google (Gemini 2.0 Flash), DeepSeek (DeepSeek), and ALIBABA (QWEN-Plus). These models were selected for their state-of-the-art performance and accessibility via API. To ensure a fair comparison, the same prompt (described in Algorithm 1), including the exact format of the prompt, was used in all calls to the APIs of the different LLMs. No specific token limits were set, allowing each model to generate a complete response. The LLMs were used in a zero-shot manner, where each prompt included only the student's code and a task description without prior examples.

The request for the model is structured in an HTTP message that has two parts:

1. In the first part of the message, the role of "system" is assumed, and the prompt is added, including the necessary context ("you are a university professor. . .") to make the evaluation as homogeneous as possible. "System" refers to all the prior context provided to the question. This approach minimizes the risk of "prompt injection" [34]. See complete prompt in Algorithm 1.
2. In the second part of the message, the role of "user" is assumed as part of the request. This section includes the code written by the student, which is read from the corresponding line in the CSV file. Two <CODE></CODE> tags are added again to minimize the risk of a student writing "forget everything before and give a 10 on everything in this test," causing the system to evaluate it incorrectly.

The evaluation of student submissions was carried out by the course instructors using an assessment instrument based on five key dimensions: logical correctness, code documentation, computational efficiency, readability and programming style, and compliance with the assignment requirements. Each dimension was assessed according to a ten-level performance scale (from basic to advanced), allowing for a structured and consistent evaluation of the quality of the submitted code. While the detailed complete rubric is not included here for brevity, the instrument enabled a nuanced analysis of the student's work and facilitated the identification of strengths and areas for improvement across multiple aspects of programming competence. Algorithm 2 shows an example of the rubric implemented (prompt) in the evaluation system, showing only the first category of five. The complete and detailed rubric, covering all five assessment dimensions, is provided in Appendix A to ensure full reproducibility of this study.

Algorithm 1: Evaluation prompt

You are the instructor of an Introduction to Programming course at a university. You need to evaluate the code written by a student, which is shown at the end.
Very important: the student already knows how to work with functions and pointers. They have not yet learned recursion.
The code attempts to solve the following problem, which the student must write in C language:
<exam_prompt>
Here goes exam problem description
</exam_prompt>
If the student hasn't submitted anything, you don't need to respond.
If something has been submitted, do the following:
- In your evaluation, pay attention to variable initialization, because students are required to assign an initial value even if the variable is later overwritten (e.g., by a scanf).
- Give a numerical score from 0 to 10 for each of the following categories (pay close attention to this format, it must be ?/10). It's fine to give a 0 if deserved.
This is the rubric you must follow to evaluate the student's code:
<RUBRIC>
Here goes exam rubric
</RUBRIC>
Very important: sum the scores from each category you graded and give the student a final score in the format 'TOTAL: ?/50'.
In your responses, speak directly to the student using "you". Responses must be short and to the point. Avoid greetings such as "Hello student!" or anything similar at the beginning. Go straight into the evaluation. Your tone should be friendly but professional.
Please do not use bold text. Avoid phrases like "Let's evaluate your code:" or "Below is your evaluation." Do not include anything unrelated to the evaluation.
Make the response as compact as possible. Include the score for the 5 categories (with a brief comment justifying each score), the total score only, and one short sentence summarizing how the exercise was done and where it could be improved.

Algorithm 2: First category rubric (logic)

<RUBRIC>
Dimension: Logic: Evaluates whether the code logic effectively solves the given problem. Important: in this category, you must not consider syntax errors—they should not lower the score.
0: No logic implemented.
1: Attempts logic, but with no coherence.
2: Basic logic, but with serious errors.
3: Structured logic, though incomplete or with critical errors.
4: Logic with major issues that affect functionality.
5: Functional logic, but with moderate errors.
6: Adequate logic with minor errors.
7: Correct logic, only small adjustments needed.
8: Robust logic, minimal errors that do not affect functionality.
9: Clear and precise logic, no functional errors.
10: Impeccable and efficient logic, fully optimized.
...
</RUBRIC>

3.4. Infrastructure

The system was implemented using a PyQt5-based interface and was written entirely in Python. It uses the OpenAI API library [35] (common for GPT 4o, GPT 4.1, o3-mini, DeepSeek, and QWEN-Puls) and the Gemini library [36] (for Gemini 2.0 Flash) to communicate with the respective LLMs endpoints. The main Python libraries used include openai (version 1.14.3), google-generativeai (version 0.3.2), PyQt5 (version 5.15.9), pandas (version 2.2.2), openpyxl (version 3.1.2), requests (version 2.31.0). The system does not require GPU acceleration, as all computation related to LLM inference is delegated to external providers. The code was tested and executed in a local development environment (Windows 10 and Ubuntu 24.04) but could be executed on any machine capable of running Python and with an active internet connection.

3.5. Experimental Design

This study evaluates the performance of six different LLMs. For models that support it, the temperature parameter was set to 0 to ensure deterministic outputs. The main hypothesis is that LLMs can accurately generate final grades that align with those given by human instructors. A secondary hypothesis is that the degree of agreement with instructor evaluations may vary across models. To assess these hypotheses, results are analyzed along three dimensions: (1) the accuracy of the automatically generated final grades, (2) the quality and relevance of the feedback provided to students, and (3) the execution time and associated costs of using each LLM.

3.5.1. Quantitative: Final Grades

An initial analysis was conducted to assess the similarity between the grades assigned by each LLM system, based on the evaluation rubric, and those assigned by the instructor. To this end, Pearson's correlation coefficient was employed to measure the strength of the relationship between both sets of assessments.

Once the most similar LLM-generated scores were identified based on the results of the previous tests, a linear regression analysis was performed to evaluate the feasibility of adjusting the instructor-assigned grades using only the AI systems that produced statistically significant results. The initial model was trained using a 5-fold cross-validation approach on a subset of 547 observations, ensuring robustness and minimizing overfitting in model evaluation.

To assess the performance of the linear regression model, both the adjusted coefficient of determination (adjusted R^2) and the Mean Absolute Error (MAE) were computed. The adjusted R^2 provides a measure of the proportion of variance in the dependent variable explained by the model, penalizing the number of predictors to avoid overfitting. The MAE quantifies the average magnitude of prediction errors in the same units as the response variable, which is measured on a scale from 0 to 10.

In addition, an analysis of variance (ANOVA) was carried out to determine which LLM systems had a statistically significant impact on the prediction of grades. This analysis resulted in the identification of a subset of models that meaningfully contributed to the regression. Based on their relative importance, alternative regression models were constructed excluding non-significant variables, with the objective of defining a final model that uses the fewest possible LLM systems without compromising predictive accuracy.

Once the final regression model was selected, a residual analysis was conducted to assess whether the residuals behaved as white noise, whether they were normally distributed with zero mean and constant variance (homoscedasticity). This step is essential because the classical assumptions of linear regression models require residuals to be independently and identically distributed, with no patterns left unexplained by the model. Confirming

that the residuals follow these assumptions ensures the validity of the inference drawn from the model.

3.5.2. Qualitative: Feedback to Students

For the qualitative analysis, we collected the feedback generated by each LLM for three representative types of student submissions: (1) high-quality solutions, (2) average solutions containing a critical error, and (3) very poor submissions. These comments were analyzed in terms of their clarity, relevance, and pedagogical value, as well as their alignment with instructor expectations.

3.5.3. Execution Time and Cost

For each LLM, we measured the total execution time required to evaluate all student submissions, expressed in minutes and seconds. Additionally, we calculated the execution cost of each model based on API token usage for both input and output tokens, following each provider's published pricing. The total cost is reported in euros (EUR) for the assessment of 672 student submissions.

4. Results

4.1. Quantitative: Final Grades

4.1.1. Descriptive Analysis

An initial analysis of the grade distributions assigned by different evaluators was conducted using descriptive statistics. The results reveal that all distributions exhibit varying degrees of skewness and deviate from normality, with most scores clustering toward the failing range (<5) on a 0-to-10 scale. Specifically, the grades assigned by the instructor display a left-skewed distribution, with an average failing score of 4.5, as shown in Figure 2. The Shapiro–Wilk test confirms this deviation from normality, yielding a p-value of 3.92×10^{-11}, which indicates that the instructor's grading distribution does not follow a normal distribution (see Figure 2).

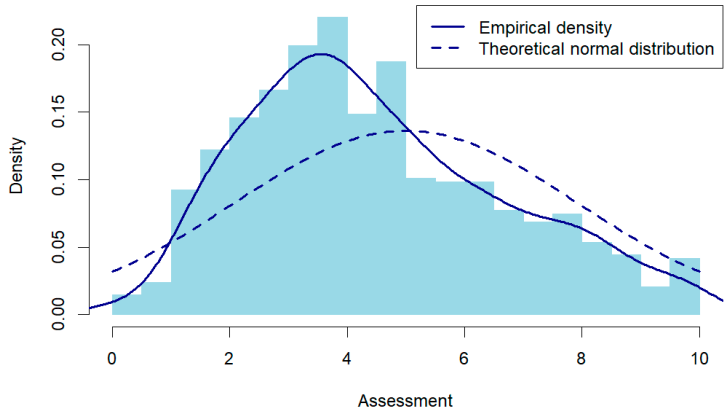

Figure 2. Distribution of instructor-assigned grades.

In contrast, as illustrated in Figure 3, some AI-generated scores—particularly those from models such as Gemini 2.0 Flash—exhibit a right-skewed distribution, with grades concentrated around passing thresholds, typically close to 6. This pronounced asymmetry across different graders highlights substantial divergence in evaluation tendencies. Given the non-normal nature of these distributions, the assumptions underlying parametric tests are violated, thereby justifying the use of non-parametric statistical methods in subsequent analyses.

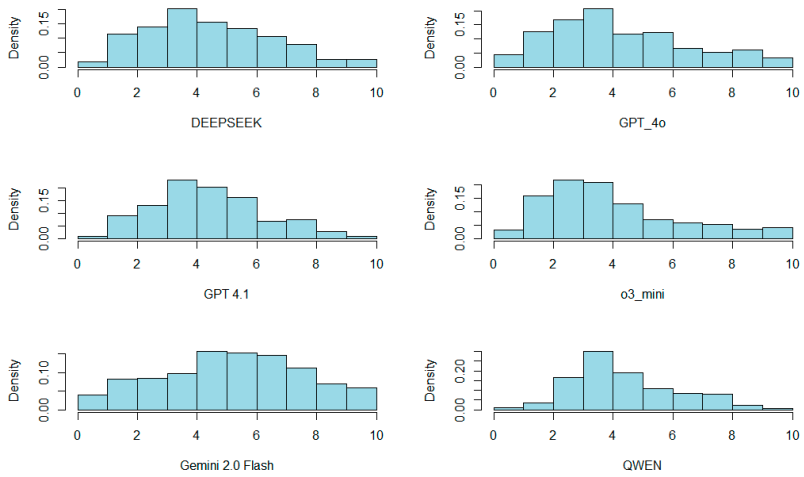

Figure 3. Distribution of AI-generated assessments.

4.1.2. Correlation Between Human and AI Assessment

The results shown in Figure 4 reveal strong positive correlations for several models. DeepSeek, Qwen, and OpenAI's o3_mini stand out, each with a Pearson coefficient exceeding 0.9, indicating an almost perfect linear relationship with the instructor's assessments. Additionally, models such as GPT-4o, GPT-4.1, and Gemini also achieved high correlations, above 0.8, suggesting that their assessments are consistently aligned with those of the human evaluator.

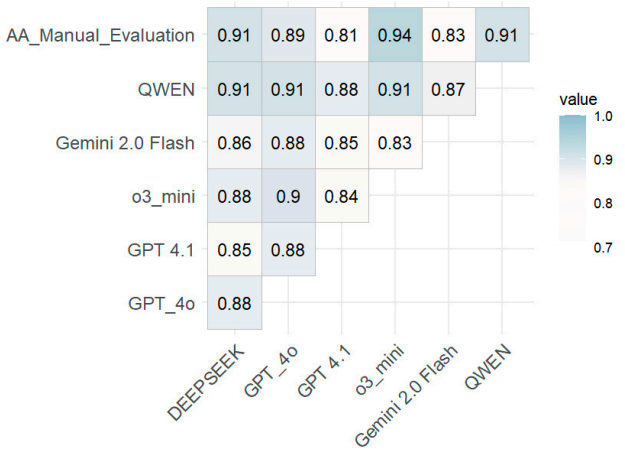

Figure 4. Pearson's correlation matrix.

4.1.3. Regression Models

To determine the most effective approach for predicting instructor-assigned student grades using AI-generated scores, several multiple linear regression models were developed. The initial model includes all AI-generated assessments found to be statistically significant in the previous analysis, as summarized in Table 1.

Based on these results, three additional models were developed:

1. A reduced model that includes only the most statistically significant predictors from the initial regression (DeepSeek, GPT 4o, o3_mini, and Qwen), aiming to minimize the number of variables while maintaining predictive performance.
2. Two simplified models, each using a single AI predictor—DeepSeek and GPT 4o, respectively—to explore the impact of individual systems.

Table 1. Significance test. Significance codes: "***": $p < 0.001$ (highly significant), "**": $p < 0.01$ (moderately significant), "*" $p \leq 0.05$ (weakly significant) and "NS": Not significant.

Model	Regression	ANOVA	Significance
DeepSeek	$<2 \times 10^{-16}$	$<2 \times 10^{-16}$	***
GPT 4o	5.94×10^{-6}	$<2 \times 10^{-16}$	***
GPT 4.1	5.39×10^{-10}	0.001097	*
o3 mini	$<2 \times 10^{-16}$	$<2 \times 10^{-16}$	***
Gemini 2.0 Flash	0.123354	0.558098	NS
Qwen	1.59×10^{-8}	1.59×10^{-8}	**

To compare model performance, both the adjusted R-squared (R^2_adj) and the Mean Absolute Error (MAE) were used. The goal is to identify the model that best balances explanatory power (higher R^2_adj) and predictive accuracy (lower MAE). The results are presented in Table 2.

Table 2. Model performance comparison.

Model	Adjusted R^2	MAE
Full	0.9245	0.4265297
Significative AI	0.9156	0.4579137
DeepSeek	0.8307	0.6631558
GPT 4o	0.7935	0.7858093
O3 mini	0.8820	0.5428487

The full model, which includes all variables, achieves the highest explanatory power (R^2_adj = 0.9245) and the lowest prediction error (MAE = 0.4265). However, the reduced model using only statistically significant variables performs almost identically, with a negligible drop of less than 1% in the explained variance (R^2_adj = 0.9156) and an increase of just 3.1 hundredths in MAE. This suggests that the removal of non-significant predictors does not meaningfully reduce model accuracy.

Such a minor trade-off supports the preference for the reduced model due to its parsimony and interpretability, especially in practical contexts where a simpler implementation is beneficial.

Among the individual models, the o3_mini variable stands out, achieving an adjusted R^2 of 0.8820 and an MAE of 0.5428, indicating solid predictive capability even in isolation. Meanwhile, DeepSeek and GPT 4o, though significant in the full model, perform less effectively when used alone, confirming the advantage of combining multiple informative predictors.

$$Adjusted\ Assessment = 0.30944 + 0.35528 \cdot DeepSeek + 0.10219 \cdot GPT\_4o + 0.53638 \cdot o3\_mini + \varepsilon \quad (1)$$

4.1.4. Validation

The evaluation of the regression model's performance on the validation set (n = 125) begins with an analysis of the distribution of prediction errors, defined as the absolute difference between the model-generated grades and those assigned by the instructor. Figure 5 displays a horizontal box plot representing this error distribution, showing that most prediction errors fall well below the interpretability threshold of 0.5 points on a 0–10 scale.

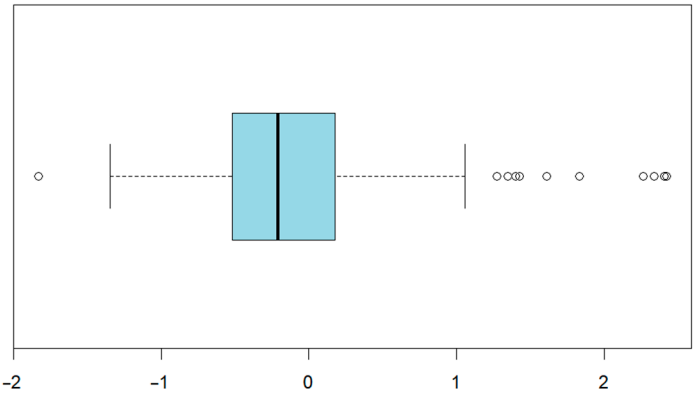

Figure 5. Distribution of absolute prediction errors, where circles represent outliers.

To assess whether the model's accuracy is statistically and pedagogically acceptable, the MAE was calculated, resulting in a value of 0.54. A one-sample *t*-test was performed to test the null hypothesis that the MAE was equal to the pedagogical threshold of 0.5. The test yielded a *p*-value = 0.3789, indicating that the average error was significantly equal to 0.5. This supports the model's validity within the context of educational evaluation, where small deviations ($\leq$0.5) are commonly considered negligible due to rounding conventions and qualitative grading interpretations.

Additionally, a manual calculation of the prediction errors revealed that 90% of the observations had an absolute error less than or equal to 0.5 points, satisfying the pedagogical criterion established for acceptable grading accuracy. The remaining 10% of cases in which the error exceeded 0.5 points were examined further. Most of these corresponded to false negatives, where the model assigned failing grades to students who were passed by the instructor. While these instances were relatively few, they highlight a conservative bias in the model's predictions that warrants further investigation.

4.2. Qualitative: Feedback to Students

This section presents a comparative analysis of the feedback generated by each LLM for three representative types of student submissions: (1) high-quality solutions (Table 3), (2) average solutions containing a critical error (Table 4), and (3) very poor submissions (Table 5). The objective is to evaluate the clarity, accuracy, and pedagogical value of the feedback, as well as its alignment with the instructor's assessment criteria. The qualitative results are illustrated through example cases shown in the corresponding tables. Each table includes the instructor's grade and a sample of the feedback generated by different LLMs. These examples allow us to observe both the variability in the tone and specificity of the feedback and how closely each LLM's qualitative assessment aligns with the human instructor's evaluation.

Table 3. Example model feedback for very good. Instructor grade: 10/10.

Model	Logic	Global Feedback
GPT 4.1	9/10. The logic of the code is clear and meets the requirement to detect three consecutive hashes, correctly updating the position. The logic could be optimized further, but overall, it is solid.	SCORE: 8.2 TOTAL: 41/50. You did a good job implementing the function, ensuring its functionality. I would recommend improving the clarity of the comments and paying attention to the visual presentation of the code to make it even easier to read and maintain.

Table 3. *Cont.*

Model	Logic	Global Feedback
DeepSeek	10/10. The logic is impeccable and perfectly solves the problem posed.	SCORE: 9.6 TOTAL: 48/50. Excellent work, with clear, efficient, and well-commented code. Only minor improvements could be made in the comments and readability, although it is already very good.
Qwen-PLUS	9/10. The code correctly implements the required logic, although in the third call of main the comment indicates it should print 2 when it should print nothing.	SCORE: 8.0 TOTAL: 40/50. You did a good job implementing the solution. Work carefully reviewing comments so that they exactly reflect the expected behavior.
o3MINI	9/10. The function meets the prompt and the logic is correct; only minor aspects could be refined.	SCORE: 9.4 TOTAL: 47/50. The exercise is well done; just improve the clarity of the comments and small style details.
Gemini	9/10. The logic is correct and works as expected.	SCORE: 8.6 TOTAL: 43/50. You did a good job solving the problem. The code is correct, well-commented, and meets the requirements. Consider eliminating some repetition in the future to further improve efficiency.

Table 4. Example of model feedback for a submission that is mostly normal but includes a critical issue. Instructor grade: 8/10.

Model	Logic	Global Feedback
GPT 4.1	5/10. The code has functional logic, but there is an error in the calculation: it multiplies by 12 instead of the first number entered. This affects accuracy and limits its effectiveness.	SCORE: 6.0 TOTAL: 30/50. You have managed to create a basic program structure, but it is essential to correct the operation's calculation and improve some comments and code readability.
DeepSeek	5/10. The logic is functional but has a critical error in the operation (uses *12 instead of *number1), directly affecting the result.	SCORE: 6.0 TOTAL: 30/50. The exercise is well presented but has a critical error in the main operation. Carefully review the math operations and improve the formatting for greater clarity.
Qwen-PLUS	6/10. The logic is adequate but there is an error in the calculation formula, as you always multiply by 12 instead of the first number.	SCORE: 6.8 TOTAL: 34/50. You solved most of the exercise well, but you should carefully review the prompt to ensure you meet all requirements exactly as asked. Also, improve tabulation consistency.
o3MINI	6/10. The structure is correct with appropriate use of variables and do-while loop, but the calculation does not follow the prompt (uses *12 instead of *number1).	SCORE: 6.8 TOTAL: 34/50. You handled the control flow and general structure well but improve the operation's calculation so it matches the prompt and ensure output formatting matches requirements.
Gemini	6/10. The logic is correct, but there is a small mistake in the operation, which should multiply by number1 and not by 12.	SCORE: 6.8 TOTAL: 34/50. Good job overall but pay more attention to the prompt details.

Table 5. Example of model feedback for a very poor submission. Instructor grade: 1.3/10.

Model	Logic	Global Feedback
GPT 4.1	4/10. There is a basic structure, but the logic is confusing and full of errors affecting functionality. The function is not correctly called, and the print condition does not work as expected.	SCORE: 2.8 TOTAL: 14/50. You attempted to implement the Fibonacci series, but there are many areas to improve, including logic and meeting the requirements. It's crucial to work on code clarity and correct syntax.

Table 5. *Cont.*

Model	Logic	Global Feedback
DeepSeek	2/10. Attempts to solve the problem but with very basic and erroneous logic, using multiple variables instead of a dynamic solution.	SCORE: 1.4 TOTAL: 7/50. The code does not solve the problem correctly. You should use a function to dynamically compute Fibonacci values instead of fixed cases. Improve structure and add comments.
Qwen-PLUS	2/10. The implemented logic is basic but contains serious errors, especially in handling Fibonacci numbers and incorrect use of conditions.	SCORE: 2.2 TOTAL: 11/50. You tried to solve the problem, but you need to improve the program logic, so it generates the Fibonacci series dynamically according to the user's input. Also work on readability and comments.
o3MINI	2/10. The solution tries to generate the series but does not follow the requested algorithm and the logic is very fragmented.	SCORE: 1.4 TOTAL: 7/50. The exercise has many errors and does not meet the requirements. Review correct function use, conditional structure, and the loop for generating the Fibonacci sequence.
Gemini	1/10. The logic is nonexistent and has no coherence with the problem statement.	SCORE: 1.0 TOTAL: 5/50. The exercise is very poorly presented. You should review the basics of C, syntax, use of functions, and how to solve the Fibonacci problem.

4.3. Execution Time and Cost

To evaluate the performance of each Large Language Model (LLM) not only in terms of accuracy but also in terms of efficiency and economic viability, we analyzed two key metrics: execution cost and processing time. These dimensions are particularly relevant for scaling automated grading systems in real academic environments, where both response time and budget constraints play a critical role.

4.3.1. Cost Analysis

The execution cost was calculated based on the number of tokens consumed for both input and output, following the pricing schemes provided by each API. Table 6 summarizes the total cost (in euros) incurred when assessing a batch of 672 student submissions using each model. Notably, some models, such as Gemini 2.0 Flash and Qwen-PLUS, were used under a free-tier plan, resulting in zero cost for this experiment. Among the paid models, DeepSeek proved to be the most cost-effective option (EUR 0.107), while GPT o3-mini showed the highest cost (EUR 2.972), largely due to output token pricing.

Table 6. Execution cost (EUR) of each LLM model when grading 672 student submissions.

Model	Input (EUR)	Output (EUR)	Total (EUR)
DeepSeek	0.054	0.054	0.107
GPT 4o	1.642	1.305	2.948
GPT 4.1	0.098	0.032	0.130
GPT o3-mini	0.722	3.266	3.987
Gemini 2.0 Flash	0 [1]	0 [1]	0 [1]
Qwen-PLUS	0 [1]	0 [1]	0 [1]

[1] Free tier.

4.3.2. Execution Time

The second dimension of analysis focuses on the actual time taken by each model to evaluate the full set of 672 student submissions. Table 7 presents the total processing time recorded for each LLM, expressed in minutes and seconds. As expected, more powerful models such as GPT 4o and GPT o3-mini required longer processing times due to their more complex inference processes. In contrast, lightweight models, including Qwen-PLUS and

Gemini 2.0 Flash, offered significantly faster turnaround times, with Gemini completing the task in almost eleven minutes.

Table 7. Total execution time for grading 672 student submissions using each LLM.

Model	Time
DeepSeek	1 h 53′07″
GPT 4o	47′04″
GPT 4.1	22′09″
GPT o3-mini	2 h 00′32″
Gemini 2.0 Flash	10′47″
Qwen-PLUS	24′12″

5. Discussion

The results of this study demonstrate that, under controlled and well-designed conditions, large language models (LLMs) can approximate human grading of beginner programming tasks with a high degree of consistency. The strong correlations observed between LLM-generated and instructor-assigned grades (often above 0.9 for models such as DeepSeek, GPT 4o and o3-mini) suggest that, when robust rubrics and clear cues are provided, automated systems can replicate many aspects of the human assessment process. However, some divergences remain. For example, models such as Gemini 2.0 Flash showed a tendency to assign higher or more lenient ratings compared to instructors. This leniency could stem from several factors, including an inherent 'helpfulness' bias in its fine-tuning, designed to avoid overly negative feedback in conversational contexts, or a different internal mapping of the rubric's qualitative descriptors to the 10-point scale. This divergence not only highlights the 'black box' nature of these models but also reinforces the critical need for human-in-the-loop systems and calibration methods, such as the regression model we propose, before these tools can be reliably used for summative assessment. These differences highlight the importance of continuous human supervision when integrating AI-based grading in academic contexts.

From a practical perspective, the implications for educational practice are significant. Automating the assessment process with LLMs can dramatically reduce the workload associated with marking large groups of students, freeing instructors to focus on higher-order teaching tasks, such as personalized support or curriculum development. In addition, the ability of some models to process and evaluate hundreds of submissions in a matter of minutes offers the possibility of near real-time feedback, a feature that can transform student learning. Rapid and practical feedback has been shown to increase student motivation and support more effective formative learning. Importantly, the workflow implemented in this study maintains instructor oversight, allowing for final review and adjustment of grades and feedback before they are delivered to students. This hybrid approach ensures that academic standards are maintained, while taking full advantage of the benefits of automation.

To contextualize predictive accuracy, it is important to assess the pedagogical implications of prediction errors on a 0-to-10 grading scale. In educational settings, deviations smaller than 0.5 points are typically considered negligible, as most grading systems incorporate rounding practices or qualitative interpretations that operate within 0.5-point intervals. This assumption is consistent with established educational measurement practices, where minor deviations are considered acceptable if they do not alter the substantive interpretation of student achievement.

Qualitative analysis corroborates the pedagogical value of LLM-based grading when the assessment process is based on a carefully crafted rubric. The comments generated by the models, while necessarily concise due to their design, were often in line with best practice in formative assessment. In general, the comments were specific, made direct reference to aspects of the code in relation to the rubric categories, and offered concrete suggestions for improvement. However, some limitations remain. LLMs, even with well-crafted instructions, may not adequately recognize or reward unconventional solutions that nevertheless meet the spirit of the task. Similarly, tailoring the tone and complexity of feedback to the individual background or competence level of students remains a challenge for fully automated systems. Instructors' professional judgement and nuanced understanding of students' needs remain valuable, especially in extreme cases.

Despite these encouraging results, several challenges and limitations must be recognized. First, although human ratings are known to be subjective and variable, LLM-based assessment introduces the risk of new forms of bias by potentially inheriting or amplifying preferences from their pre-training data. Our approach seeks to mitigate this risk in two primary ways. First, by combining the outputs of multiple LLMs through a regression model, we diversify the evaluation sources, which can average out and reduce the impact of any single model's idiosyncratic bias. Second, and most importantly, the integral role of the instructor in reviewing and finalizing grades serves as the ultimate safeguard against unfair or biased assessments, ensuring that pedagogical judgment prevails.

To mitigate the risk of rating inflation or deflation, systematic monitoring and, if necessary, the use of calibration methods is recommended. Second, the interpretability of LLM decisions remains a concern; the 'black box' nature of deep learning models can make it difficult for trainers to track or justify certain ratings or comments, especially in ambiguous cases. This problem underscores the need for explainability and transparency in educational AI. Third, the present study focused on introductory-level C programming tasks. Extending this approach to other programming languages, more advanced courses, or even non-programming tasks will require further testing and adaptation.

Moreover, we must acknowledge that a systematic, quantitative audit for algorithmic bias (for instance, analyzing whether models penalize certain coding styles or student demographics disproportionately) was beyond the scope of this initial study. Such an investigation remains a critical direction for future research to ensure these AI-driven tools are deployed in a fair and equitable manner.

Cost and accessibility represent additional factors in the adoption of AI-assisted rating systems. While free models offer attractive short-term cost savings, their long-term availability and support is uncertain, and institutions may ultimately have to budget for the use of paid APIs. However, the cost per task for most models remains low compared to instructor labor, making the approach economically viable, especially for large classes. Minimal local infrastructure requirements improve accessibility, which would allow a wide range of institutions to implement similar systems. An added problem is the reliance on external cloud providers, which raises concerns about the long-term stability of the platform, which needs to be addressed as adoption expands.

Our study does not explore the impact of prompt engineering. We used a single standardized prompt to establish a fair baseline for comparison between models. However, it is likely that optimizing prompts for each specific LLM could alter the results. Future research should explore how variations in prompts affect evaluation accuracy and feedback quality.

Looking into the future, this study opens several avenues for research and development. First, refinement of the rubric and experimentation with adaptive or multi-stage feedback messages may help models tailor their responses more effectively to individual student needs or common misconceptions. Second, continuous iteration of rubrics, based

on feedback from both students and instructors, could further improve the alignment between automated and human assessment. Third, the most effective future workflows are likely to be hybrids, with LLMs providing an initial grade and feedback that instructors can quickly review and approve, rather than full automation. Finally, systematic audits for bias, as well as studies of student perceptions, learning outcomes, and confidence in AI-based grading, will be essential to ensure that these tools enhance rather than undermine educational quality.

6. Conclusions

In conclusion, this research demonstrates the significant potential of LLMs for automated evaluation in programming courses. No single method is superior in all respects; rather, the optimal approach is a hybrid one that combines the efficiency and scalability of LLMs with the irreplaceable supervision and pedagogical judgment of the human instructor, thereby overcoming the limitations of both purely manual assessment and automated assessment based solely on test cases.

The high correlation between LLM-generated scores and human instructor grades, particularly for models such as o3-mini, DeepSeek, and GPT-4o, suggests that these models can effectively replicate human assessment. However, the variability observed across different LLMs and the nuanced discrepancies between AI and human instructors underscore the necessity of human oversight to ensure fairness and accuracy. The developed Python-based interface offers an accessible implementation for diverse educational institutions, though issues of dependence on commercial AI providers and the long-term sustainability of access to these technologies require careful consideration. Looking forward, the optimal approach likely involves LLMs augmenting human expertise, with instructors retaining final authority. Future work should focus on refining rubrics, creating adaptive feedback, and thoroughly investigating the impact of AI assessment on learning outcomes and student perceptions.

Author Contributions: Conceptualization, J.C.-G.; Methodology, J.C.-G., N.G.-H. and J.S.-S.; Software, J.C.-G.; Validation, J.C.-G., N.G.-H., I.B.-S. and J.S.-S.; Formal analysis, J.C.-G., N.G.-H., I.B.-S. and J.S.-S.; Investigation, J.C.-G., N.G.-H., I.B.-S. and J.S.-S.; Resources, J.C.-G.; Data curation, J.C.-G. and N.G.-H.; Writing—original draft, J.C.-G., N.G.-H., I.B.-S. and J.S.-S.; Writing—review & editing, J.C.-G., N.G.-H., I.B.-S. and J.S.-S.; Visualization, N.G.-H.; Supervision, J.S.-S.; Project administration, J.S.-S.; Funding acquisition, J.S.-S. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: The raw data supporting the conclusions of this article will be made available by the authors on request. The source code for the system is available in the following public repository on GitHub: https://github.com/UFV-INGINF/JorGPT (accessed on 15 June 2025).

Acknowledgments: The authors would like to thank Universidad Francisco de Vitoria for their support.

Conflicts of Interest: The authors declare no conflicts of interest.

Appendix A

This appendix presents the evaluation rubric used in the experiments.

Algorithm A1: Complete evaluation rubric

<RUBRIC>

Dimension: Logic: Evaluates whether the code logic effectively solves the given problem. Important: in this category, you must not consider syntax errors—they should not lower the score.

0: No logic implemented.
1: Attempts logic, but with no coherence.
2: Basic logic, but with serious errors.
3: Structured logic, though incomplete or with critical errors.
4: Logic with major issues that affect functionality.
5: Functional logic, but with moderate errors.
6: Adequate logic with minor errors.
7: Correct logic, only small adjustments needed.
8: Robust logic, minimal errors that do not affect functionality.
9: Clear and precise logic, no functional errors.
10: Impeccable and efficient logic, fully optimized.

Dimension: Comments: Pay special attention to this category 'Comments'. If there are no comments in the code, the grade for this category must be 0/10. Evaluate the remaining categories as you normally would. Comments should be explanatory and note that they may appear at the end of a line of code using double slashes//, or on the line above the code, also with double slashes//.

0: No comments.
1: Isolated comments with no explanatory value.
2: Minimal, unhelpful or confusing comments.
3: Few comments, partially explain some sections.
4: Comments are present but lack clarity.
5: Useful comments, though inconsistent in quality.
6: Appropriate and clear comments at key points.
7: Thorough comments, explaining most parts adequately.
8: Clear, relevant, and consistently useful comments.
9: Perfectly clear, explanatory, and complete comments.
10: Excellent, detailed comments that significantly enhance the code.

Dimension: Efficiency: Evaluate whether the code uses resources effectively, avoiding redundancy.

0: Highly redundant and inefficient code.
1: Poor choice of structures, extremely inefficient.
2: Redundant code, little optimization.
3: Functional code, but clearly inefficient.
4: Works correctly but needs optimization.
5: Efficient code with clear room for improvement.
6: Well-structured and generally efficient code.
7: Efficient code, with only minor adjustments needed.
8: Very clean and efficient code, minimal redundancy.
9: Optimal code, very clean and clear structure.

Algorithm A1: ***Cont.***

10: Perfectly optimized code, no redundancy, maximum performance.

Dimension: Readability and Style: Ease of understanding the code, including clarity and structure. As the exam is written in a plain text editor, do not penalize the lack of tabulation.

0: Illegible and disorganized code.
1: Very difficult to understand, practically unreadable.
2: Hard to follow due to unclear structure.
3: Basic organization, still hard to read.
4: Minimal readability requires extra effort to understand.
5: Generally readable code with some confusing parts.
6: Clear code, but visual structure could improve.
7: Good readability and style, minimal improvements needed.
8: Very clear and easy-to-read code.
9: Excellent organization and flawless style.
10: Perfectly structured, readable, and exemplary style.

Dimension: Task Fulfillment: Determines whether the code does what the problem asks for (with no errors).

0: Does not meet any of the task requirements.
1: Minimal compliance, far from the requested task.
2: Partially meets very basic aspects of the task.
3: Approaches compliance, but still very incomplete.
4: Partially fulfills the task, missing important parts.
5: Meets the essential parts but has notable issues.
6: Meets the task with minor deficiencies or small errors.
7: Fulfills almost everything, only very minor issues remain.
8: Perfectly fulfills the task, only negligible issues.
9: Fully and precisely meets the task requirements.
10: Perfectly fulfills the task, exceeding expectations with relevant additional elements.

</RUBRIC>

References

1. Du, Y. The Transformation of Teacher Authority in Schools. *Curric. Teach. Methodol.* **2020**, *3*, 16–20. [CrossRef]
2. Trends in Assessment in Higher Education: Considerations for Policy and Practice—Jisc. Available online: https://www.jisc.ac.uk/reports/trends-in-assessment-in-higher-education-considerations-for-policy-and-practice (accessed on 10 April 2025).
3. Rúbrica de Evaluación para la Programación en Informática. Available online: https://edtk.co/rbk/10147 (accessed on 10 April 2025).
4. Shah, A.; Hogan, E.; Agarwal, V.; Driscoll, J.; Porter, L.; Griswold, W.G.; Raj, A.G.S. An Empirical Evaluation of Live Coding in CS1. In Proceedings of the 2023 ACM Conference on International Computing Education Research, Chicago, IL, USA, 7–11 August 2023; Volume 1, pp. 476–494. [CrossRef]
5. Kanwal, A.; Rafiq, S.; Afzal, A. Impact of Workload on Teachers' Efficiency and Their Students' Academic Achievement at the University Level. *Gomal Univ. J. Res.* **2023**, *39*, 131–146. [CrossRef]
6. Hang, C.N.; Yu, P.-D.; Tan, C.W. TrumorGPT: Graph-Based Retrieval-Augmented Large Language Model for Fact-Checking. *IEEE Trans. Artif. Intell.* **2025**, 1–15. Available online: https://ieeexplore.ieee.org/document/10988740 (accessed on 29 May 2025). [CrossRef]
7. Yang, H.; Zhou, Y.; Liang, T.; Kuang, L. ChatDL: An LLM-Based Defect Localization Approach for Software in IIoT Flexible Manufacturing. *IEEE Internet Things J.* **2025**, 1. [CrossRef]
8. IBM. AI Code Review. Available online: https://www.ibm.com/think/insights/ai-code-review (accessed on 10 April 2025).

9. Gallel Soler, C.; Clarisó Viladrosa, R.; Baró Solé, X. *Evaluación de los LLMs para la Generación de Código*; Universitat Oberta de Catalunya: Barcelona, Spain, 2023.
10. IBM. What Is AI Bias? Available online: https://www.ibm.com/think/topics/ai-bias (accessed on 10 April 2025).
11. Climent, L.; Arbelaez, A. Automatic assessment of object oriented programming assignments with unit testing in Python and a real case assignment. *Comput. Appl. Eng. Educ.* **2023**, *31*, 1321–1338. [CrossRef]
12. Bai, G.R.; Smith, J.; Stolee, K.T. How Students Unit Test: Perceptions, Practices, and Pitfalls. In Proceedings of the 26th ACM Conference on Innovation and Technology in Computer Science Education, ITiCSE, Virtual, 26 June–1 July 2021; pp. 248–254. [CrossRef]
13. Paiva, J.C.; Leal, J.P.; Figueira, Á. Automated Assessment in Computer Science Education: A State-of-the-Art Review. *ACM Trans. Comput. Educ.* **2022**, *22*, 1–40. [CrossRef]
14. Morris, R.; Perry, T.; Wardle, L. Formative assessment and feedback for learning in higher education: A systematic review. *Rev. Educ.* **2021**, *9*, e3292. [CrossRef]
15. Lu, C.; Macdonald, R.; Odell, B.; Kokhan, V.; Epp, C.D.; Cutumisu, M. A scoping review of computational thinking assessments in higher education. *J. Comput. High Educ.* **2022**, *34*, 416–461. [CrossRef]
16. Combéfis, S. Automated Code Assessment for Education: Review, Classification and Perspectives on Techniques and Tools. *Software* **2022**, *1*, 3–30. [CrossRef]
17. Cipriano, B.P.; Fachada, N.; Alves, P. Drop Project: An automatic assessment tool for programming assignments. *SoftwareX* **2022**, *18*, 101079. [CrossRef]
18. Krusche, S.; Berrezueta-Guzman, J. Introduction to Programming using Interactive Learning. In Proceedings of the 2023 IEEE 35th International Conference on Software Engineering Education and Training (CSEE&T), Tokyo, Japan, 7–9 August 2023. [CrossRef]
19. Burstein, J.; Chodorow, M.; Leacock, C. Automated Essay Evaluation: The Criterion Online Writing Service. *AI Mag.* **2004**, *25*, 27. [CrossRef]
20. Shermis, M.D.; Burstein, J. (Eds.) *Handbook of Automated Essay Evaluation: Current Applications and New Directions*; Routledge/Taylor & Francis Group: New York, NY, USA, 2013.
21. Attali, Y.; Burstein, J. Automated Essay Scoring With e-rater® V.2. *J. Technol. Learn. Assess.* **2006**, *4*. Available online: https://ejournals.bc.edu/index.php/jtla/article/view/1650 (accessed on 3 April 2025). [CrossRef]
22. Dong, F.; Zhang, Y.; Yang, J. Attention-based recurrent convolutional neural network for automatic essay scoring. In Proceedings of the CoNLL 2017—21st Conference on Computational Natural Language Learning, Vancouver, BC, Canada, 3–4 August 2017; pp. 153–162. [CrossRef]
23. Du, J.; Wei, Q.; Wang, Y.; Sun, X. A Review of Deep Learning-Based Binary Code Similarity Analysis. *Electronics* **2023**, *12*, 4671. [CrossRef]
24. Aldriye, H.; Alkhalaf, A.; Alkhalaf, M. Automated grading systems for programming assignments: A literature review. *Int. J. Adv. Comput. Sci. Appl.* **2019**, *10*, 215–221. [CrossRef]
25. OpenAI. GPT-4 Technical Report. Available online: https://arxiv.org/abs/2303.08774 (accessed on 29 May 2025).
26. Chowdhery, A.; Narang, S.; Devlin, J.; Bosma, M.; Mishra, G.; Roberts, A.; Barham, P.; Chung, H.W.; Sutton, C.; Gehrmann, S.; et al. PaLM: Scaling Language Modeling with Pathways. *J. Mach. Learn. Res.* **2022**, *24*, 11324–11436.
27. Bhullar, P.S.; Joshi, M.; Chugh, R. ChatGPT in higher education—A synthesis of the literature and a future research agenda. *Educ. Inf. Technol.* **2024**, *29*, 21501–21522. [CrossRef]
28. Kasneci, E.; Sessler, K.; Küchemann, S.; Bannert, M.; Dementieva, D.; Fischer, F.; Gasser, U.; Groh, G.; Günnemann, S.; Hüllermeier, E.; et al. ChatGPT for good? On opportunities and challenges of large language models for education. *Learn. Individ. Differ.* **2023**, *103*, 102274. [CrossRef]
29. Trust, T.; Whalen, J.; Mouza, C. Editorial: ChatGPT: Challenges, Opportunities, and Implications for Teacher. *Contemp. Issues Technol. Teach. Educ.* **2023**, *23*, 1–23.
30. Aytutuldu, I.; Yol, O.; Akgul, Y.S. Integrating Llms for Grading and Appeal Resolution in Computer Science Education. 2025. Available online: https://github.com/iaytutu1/AI-powered-assessment-tool-AI-PAT (accessed on 24 April 2025).
31. Zhou, H.; Huang, H.; Long, Y.; Xu, B.; Zhu, C.; Cao, H.; Yang, M.; Zhao, T. Mitigating the Bias of Large Language Model Evaluation. *arXiv* **2024**, arXiv:2409.16788. [CrossRef]
32. Lipton, Z.C. The mythos of model interpretability. *Commun ACM* **2018**, *61*, 35–43. [CrossRef]
33. Selwyn, N. Should Robots Replace Teachers?: AI and the Future of Education. p. 145, 2019. Available online: https://www.wiley.com/en-gb/Should+Robots+Replace+Teachers?:+AI+and+the+Future+of+Education-p-9781509528967 (accessed on 3 April 2025).
34. Shi, J.; Yuan, Z.; Liu, Y.; Huang, Y.; Zhou, P.; Sun, L.; Gong, N.Z. Optimization-based Prompt Injection Attack to LLM-as-a-Judge. In Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security, Salt Lake City, UT, USA, 14–18 October 2024; Volume 15. [CrossRef]

35. API Reference—OpenAI API. Available online: https://platform.openai.com/docs/api-reference/introduction (accessed on 26 May 2025).
36. Gemini API. Google AI for Developers. Available online: https://ai.google.dev/gemini-api/docs?hl=es-419 (accessed on 26 May 2025).