Assessing Large Language Models for Automated Feedback Generation in Learning Programming Problem Solving

Priscylla Silva PRISCYLLA.SILVA@USP.BR
Universidade de São Paulo and Instituto Federal de Alagoas

Evandro Costa EVANDRO@IC.UFAL.BR
Universidade Federal de Alagoas

Abstract

Providing effective feedback is important for student learning in programming problem-solving. In this sense, Large Language Models (LLMs) have emerged as potential tools to automate feedback generation. However, their reliability and ability to identify reasoning errors in student code remain not well understood. This study evaluates the performance of four LLMs (GPT-4o, GPT-4o mini, GPT-4-Turbo, and Gemini-1.5-pro) on a benchmark dataset of 45 student solutions. We assessed the models' capacity to provide accurate and insightful feedback, particularly in identifying reasoning mistakes. Our analysis reveals that 63% of feedback hints were accurate and complete, while 37% contained mistakes, including incorrect line identification, flawed explanations, or hallucinated issues. These findings highlight the potential and limitations of LLMs in programming education and underscore the need for improvements to enhance reliability and minimize risks in educational applications.

Keywords: Programming Education, Large Language Models, Automated Feedback, Automated Assessment

1. Introduction

Despite its foundational role in computing education, computer programming remains a difficult subject for many learners. Programming courses often enroll large numbers of students, making it difficult for instructors to offer timely and personalized feedback. This feedback is crucial for helping learners understand complex concepts, correct misconceptions, and develop problem-solving skills. However, generating such feedback for programming assignments are particularly demanding. Instructors must evaluate whether a solution meets the problem's requirements, understand the student's thought process, identify errors, and communicate guidance to foster reflection and learning (Silva et al., 2019).

Automated feedback systems have emerged as a solution to address the challenges of assessing student submissions. Traditional tools typically rely on predefined rules or rigid evaluation frameworks (Deeva et al., 2021). While these methods can be effective in specific contexts, they lack the flexibility to provide individualized and context-aware feedback (Maier and Klotz, 2022). Recent studies have demonstrated that large language models (LLMs) can be used to overcome this limitation by providing natural student feedback that adapts to each student's specific needs (Kiesler et al., 2023).

However, using LLMs in educational contexts raises significant concerns. Tyen et al. (2024) show that LLMs struggle with reasoning tasks, such as accurately following the step-by-step resolution of arithmetic expressions. These models show less than 50% accuracy in

both arithmetic and logical deduction tasks, particularly in identifying and correcting errors, as they often fail to indicate the specific step where a mistake occurred. Furthermore, large language models (LLMs) are susceptible to generating hallucinations or providing misleading information, which may frustrate students and interfere with the learning process (Jukiewicz, 2024). Such inaccuracies are particularly concerning in education, where trust and reliability are paramount.

In this work, we address these challenges by providing a systematic evaluation of four different LLMs (GPT-4o, GPT-4o-mini, GPT-4-Turbo, and Gemini-1.5-pro) on their ability to generate accurate and meaningful feedback for programming assignments. Unlike prior studies that typically focus on a single LLM, Our comparative analysis highlights the different capabilities and limitations of multiple LLMs. To ensure reproducibility, we contribute a benchmark dataset based on real-world student submissions from introductory programming courses, making it publicly available to foster further research in this area.

In summary, the main contributions of this work are:

1. A evaluation of four LLMs in generating feedback for programming assignments, focusing on their ability to identify and explain student mistakes;

2. A publicly available benchmark dataset of annotated student code submissions, facilitating future research on automated feedback systems.

2. Related Work

Evaluating Feedback Generated by LLMs in Programming Tasks. Pankiewicz and Baker (2023) used GPT-3.5 to provide hints to students upon request during programming tasks when errors such as compilation failures, runtime issues, or unit test failures were identified. Their approach used prompts that included the problem description, student code, and error details. Feedback usefulness was evaluated based on student opinions using a Likert scale. However, the study did not assess whether the feedback correctly aligned with the problem requirements or addressed issues such as hallucinations or logical inconsistencies.

Azaiz et al. (2024) examined GPT-4 Turbo for feedback generation on 55 solutions to two programming assignments. They reported feedback accuracy rates of 75% to 95% in determining the correctness of solutions but noted that 22% of the feedback was inconsistent or incorrect. Similarly, Roest et al. (2024) evaluated GPT-3.5 Turbo for real-time hint generation during students' problem-solving processes. Experts reviewed 48 hints and found that 33% contained misleading information. Jacobs and Jaschke (2024) employed GPT-4 to generate feedback using task specifications, student code, compiler output, and unit test results. Expert evaluation of 137 feedbacks revealed that 12% contained incorrect suggestions, and 6% exhibited hallucinations.

These studies underscore the challenges in ensuring the accuracy and reliability of LLM-generated feedback. Although the proportion of issues in feedback (ranging from 6% to 33%) may appear small, in the context of education, even a small rate of misleading feedback can negatively affect students' learning, leading to misunderstandings, frustration, and loss of trust in the system. While these studies focus on evaluating individual models and their ability to generate feedback, our research goes further by conducting a systematic comparison of multiple LLMs.

Challenges in Evaluating Reasoning with LLMs. Detecting reasoning mistakes in student code requires understanding the underlying logic and intent of the solution, going beyond surface-level syntax or test failures. Recent research has shown that LLMs generally perform poorly in identifying logical mistakes compared to their ability to correct them. Tyen et al. (2024) demonstrated that several LLMs struggle with logical deduction, arithmetic expressions, and word-ordering tasks, often failing to pinpoint where reasoning mistakes occur. Similarly, Xia et al. (2024) observed similar limitations in mathematical reasoning tasks, where LLMs frequently misinterpret problem-solving logic.

3. Benchmark

We created a benchmark dataset based on real-world student submissions to evaluate LLMs' performance in providing feedback for programming tasks. This dataset comprises 45 Python solutions to 5 introductory programming assignments collected from an online system used in programming courses. These 45 solutions were submitted by 5 students, with some students providing multiple solutions for the same programming assignment. This reflects a range of solution attempts and iterative improvements, capturing diverse solution strategies and common errors. Each assignment was designed to test foundational programming concepts such as loops, conditionals, and functions.

Dataset Construction. The dataset was generated from student submissions to an automated grading system, which determined the correctness of each solution based on problem requirements and test cases. The students' solutions were written in Python 3.11. Table 1 shows an overview of the dataset collected. This dataset was used to assess LLMs' feedback and evaluate their ability to identify mistakes and suggest improvements and corrections.

Table 1: Distribution of students' solutions across assignments and correctness based on the submission system.

Assignment	**Total Solutions**	**Correct**	**Incorrect**
Area of a Circle	14	05	09
Simple Sum	06	04	02
T-Shirts	10	02	08
Huaauhahhuahau	08	04	04
Grandpa's Balance	07	04	03
Total	45	19	26

Feedback Generation. Feedback was generated using four LLMs: GPT-4o (version: 2024-08-06), GPT-4o-mini (version: 2024-07-18), GPT-4-Turbo (version: 2024-04-09), and Gemini-1.5-pro. Each model was prompted using the same template to ensure consistency in feedback generation (see the prompt template in Figure 1). The models were executed with a temperature setting of 0 (zero) to try to ensure consistent output results. The prompt explicitly instructed the models to act as programming teachers, assess the correctness of

the code, and generate JSON-formatted feedback. Feedback included a binary indicator (*is_correct*) and a list of hints, with each hint specifying:

- The line number of the issue,
- The code line containing the issue,
- A concise explanation of the problem.

```
I want you to act as a programming teacher that helps students to solve a
programming assignment in Python 3.11.
Review the student's code for the following problem:

Problem: {problem description}

Student Code: {student' solution}

Identify if the code executes correctly and fulfills the problem's
requirements. If not, provide a JSON object with:
    - "is_correct": A boolean indicating whether the code is correct.
    - "hints": An array of objects, each with:
      - "line_number": The line number of the issue.
      - "code_line": The code line with the issue.
      - "hint": A short explanation of what is wrong.

Example output:
    {
      "is_correct": false,
      "hints": [
        {
          "line_number": 3,
          "code_line": "x = x + 2",
          "hint": "The variable 'x' is used before being initialized."
        },
        {
          "line_number": 5,
          "code_line": "if x > 0:",
          "hint": "The loop condition does not account for edge cases."
        }
      ]
    }

If the code is correct, return:
{
  "is_correct": true,
  "hints": []
}
```

Figure 1: Prompt used to ask for feedback.

Feedback Annotation. Two teaching assistants annotated the feedback generated by the models. The feedback was assessed in two stages:

1. **Automatic Annotation**: The correctness of the solutions was determined using the results from the automated submission system.
2. **Human Annotation**: Teaching assistants evaluated the feedback and organized it into five categories. Disagreement cases were resolved through discussion and consensus.

Each hint was categorized individually, following an adaptation of the framework by Hellas et al. (2023). In our categorization, any error in the hint message makes it incorrect, even if other parts are accurate. The categories are defined as follows:

- **Accurate and Complete (AC)**: The feedback correctly identifies the appropriate line of code and explains the issue and/or how to resolve it. This is considered the ideal type of feedback, as it effectively supports students in identifying and understanding their mistakes.
- **Accurate Line Only (ALO)**: The feedback correctly identifies the line containing the issue but provides an incorrect or misleading explanation of the problem. Unlike False Positive (FP), the line identified does indeed contain a mistake, but the model misunderstands the nature of the issue.
- **Accurate Issue Only (AIO)**: The feedback describes the right issue but does not indicate the correct line of code. This misalignment can confuse students, as the explanation may not appear to relate to the line they are working on.
- **False Positive (FP)**: The feedback incorrectly identifies an issue where none exists. There is no actual error on the identified line, and the feedback message is entirely fake (hallucinated). This type of feedback can undermine student trust in the system.
- **Misleading Suggestions** (MS): The feedback points out a real error in the code, but both the line and the suggested fix are incorrect. While related to AIO, MS adds an additional layer of confusion by including a flawed or misleading resolution.

Table 2 displays the categorization of the hints produced by each model, indicating the frequency of each category. The analysis focuses only on true negatives, which means that the dataset includes only the feedback for solutions that the models accurately identified as incorrect.

Table 2: Categorization of hints generated by each model.

Model	**Categories**					
	AC	**ALO**	**AIO**	**FP**	**MS**	**Total**
GPT-4o-mini	36	02	06	03	02	49
GPT-4o	28	10	03	08	00	49
GPT-4-Turbo	33	06	02	03	00	44
Gemini-1.5-pro	23	09	07	09	00	48
Total	120	27	18	23	02	190

4. Evaluation of Model Performance

To assess the ability of LLMs to evaluate student code, we use our benchmark dataset to investigate two key research questions:

RQ1. How accurately do LLMs determine whether a student's solution fulfills the requirements of a programming task?

RQ2. What are the common types of errors in student code that LLMs fail to identify?

4.1. Results for RQ1: Accuracy in Correctness Evaluation

The models performed similarly in evaluating the correctness of students' code (Figure 2 shows a visualization of the confusion matrices for each model). The accuracy metrics for each model are as follows: GPT-4o-mini: 84.44%, Gemini-1.5-pro: 86.67%, GPT-4o: 88.89%, and GPT-4-Turbo: 88.89%.

GPT-4o-mini misclassified seven correct solutions as incorrect. These errors were distributed across three tasks: four from the *Simple Sum* assignment, two from the *T-shirt* assignment, and one from *Grandpa's Balance*. In the *Simple Sum* task, the model incorrectly claimed the solutions did not meet the required output format, despite their correctness. Similarly, in the *T-shirt* and *Grandpa's Balance* assignments, the model misinterpreted the students' logical reasoning, falsely identifying errors in the code.

GPT-4o exhibited similar misclassification patterns, with five correct solutions identified as incorrect. These were a subset of the errors made by GPT-4o-mini, including three *Simple Sum* solutions and two *T-shirt* solutions. GPT-4o successfully identified that one solution from *Simple Sum* met the output requirements, which GPT-4o-mini failed to recognize. However, the model also struggled to interpret logical reasoning in the *T-shirt* task.

GPT-4-Turbo demonstrated comparable accuracy to GPT-4o, misclassifying four correct solutions as incorrect. It improved over GPT-4o-mini in the *Simple Sum* assignment, reducing errors to two cases. However, like the others, it struggled with the logical reasoning in two *T-shirt* assignment solutions.

Gemini-1.5-pro exhibited a similar pattern of errors as the GPT models. The misclassified solutions were a subset of those incorrectly labeled by GPT-4o and GPT-4o-mini. Gemini and GPT-4-Turbo were the only ones to classify an incorrect solution as correct incorrectly.

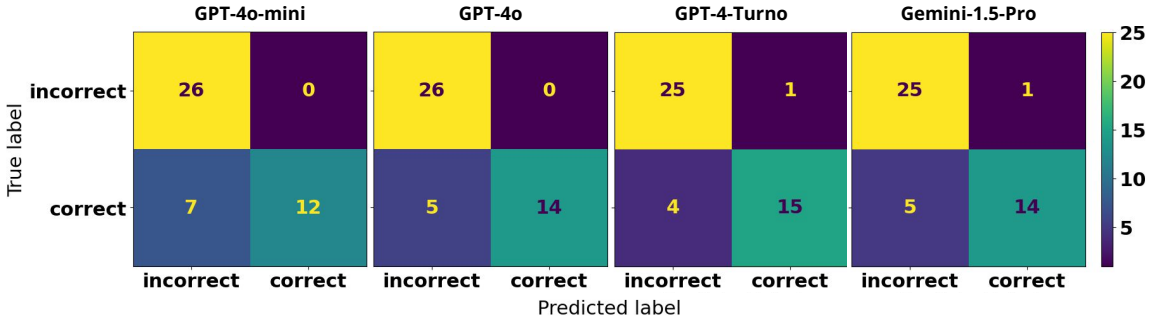

Figure 2: Confusion matrices for GPT-4o, GPT-4o-mini, and GPT-4-Turbo, showing the models' performance in classifying correct and incorrect student solutions.

4.2. Results for RQ2: Error Detection and Feedback Quality

To evaluate the quality of feedback generated by the models, we categorized hints associated with correctly identified incorrect solutions into five categories: Accurate and Complete

(AC), Accurate Line Only (ALO), Accurate Issue Only (AIO), False Positive (FP), and Misleading Suggestions (MS). Figure 3 illustrates the distribution of feedback categories across models.

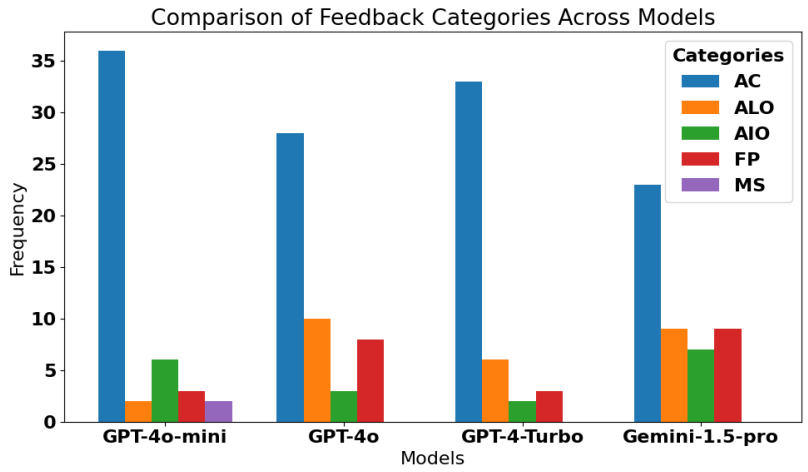

Figure 3: Comparison of the frequency of feedback categories generated by each model.

GPT-4o-mini generated the highest number of *Accurate and Complete* hints, followed closely by GPT-4-Turbo. These results indicate the strength of these models in providing both the correct line and the proper explanation of the mistake. Furthermore, GPT-4o and Gemini-1.5-pro generated more *Accurate Line Only* hints, suggesting a tendency to identify the line containing the issue but not adequately explaining the problem or assisting the student in fixing it. Across all models, the frequency of *False Positives* was relatively low, occurring in approximately 12% of cases. These errors were more frequent in Gemini-1.5-pro and GPT-4o. Similarly, *Misleading Suggestions* were rare, with only two instances reported for GPT-4o-mini.

5. Limitations and Future Directions

This study has potential limitations. First, the benchmark dataset comprises 45 solutions from 5 students across five programming assignments. Although it captures diverse solution strategies and common errors, the dataset's small size and limited participant pool restrict the generalizability of our findings. Furthermore, imbalances in the number of submissions across assignments may introduce bias in the evaluation results. We plan to expand the dataset with more participants, diverse assignments, and augmented data to address these issues.

Second, the prompt design may have impacted the models' performance. In future studies, we will explore alternative prompt designs, such as separating tasks into individual prompts or using dynamic, task-specific prompts. This could provide a more accurate evaluation of LLM capabilities.

Finally, our findings are based on the evaluation of four specific LLMs. While these models represent state-of-the-art capabilities, the results may not generalize to other mod-

els or future iterations. Additionally, while human annotations were validated to ensure consistency, they may still introduce an element of subjectivity.

6. Discussion and Conclusion

In this work, we evaluate the ability of large language models (LLMs) to provide feedback on programming tasks. We introduce a benchmark dataset of student submissions in introductory programming courses and use it to analyze and compare the performance of GPT-4o, GPT-4o-mini, GPT-4-Turbo, and Gemini-1.5-pro. Our study categorizes feedback generated by these models into five categories, highlighting their strengths and limitations in providing proper feedback. The results show that while these LLMs are effective in generating accurate feedback in many cases, they also exhibit significant challenges, including hallucinated errors, misleading suggestions, and struggles to interpret student logic.

Overall, 63% of the feedback hints generated by the models were totally correct, meaning they accurately identified the problematic line and provided an appropriate explanation. However, 37% of the feedback contained some issues, such as pointing to the wrong line, providing an incorrect hint message, or hallucinating non-existent errors.

Our findings reveal that although LLMs perform well in detecting syntactic and surface-level issues, they often fail to capture deeper reasoning mistakes in student code. Additionally, the frequency of hallucinations and false positives highlights the need for further refinement of these models to minimize their risks in educational settings. The benchmark, prompts, and supplementary materials can be found at: https://github.com/priscylla/Assessing-LMMs-for-Feedback-Generation.

References

Imen Azaiz, Natalie Kiesler, and Sven Strickroth. Feedback-generation for programming exercises with gpt-4. In *Proceedings of the 2024 on Innovation and Technology in Computer Science Education V. 1*, ITiCSE 2024, page 31–37, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400706004. doi: 10.1145/3649217.3653594. URL https://doi.org/10.1145/3649217.3653594.

Galina Deeva, Daria Bogdanova, Estefanía Serral, Monique Snoeck, and Jochen De Weerdt. A review of automated feedback systems for learners: Classification framework, challenges and opportunities. *Computers & Education*, 162:104094, 2021. ISSN 0360-1315. doi: https://doi.org/10.1016/j.compedu.2020.104094. URL https://www.sciencedirect.com/science/article/pii/S036013152030292X.

Arto Hellas, Juho Leinonen, Sami Sarsa, Charles Koutcheme, Lilja Kujanpää, and Juha Sorva. Exploring the responses of large language models to beginner programmers' help requests. In *Proceedings of the 2023 ACM Conference on International Computing Education Research - Volume 1*, ICER '23, page 93–105, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 9781450399760. doi: 10.1145/3568813.3600139. URL https://doi.org/10.1145/3568813.3600139.

Sven Jacobs and Steffen Jaschke. Evaluating the application of large language models to generate feedback in programming education. In *2024 IEEE Global Engineering Education Conference (EDUCON)*, pages 1–5, 2024. doi: 10.1109/EDUCON60312.2024.10578838.

Marcin Jukiewicz. The future of grading programming assignments in education: The role of chatgpt in automating the assessment and feedback process. *Thinking Skills and Creativity*, 52:101522, 2024. ISSN 1871-1871. doi: https://doi.org/10.1016/j.tsc.2024.101522. URL https://www.sciencedirect.com/science/article/pii/S1871187124000609.

Natalie Kiesler, Dominic Lohr, and Hieke Keuning. Exploring the potential of large language models to generate formative programming feedback. In *2023 IEEE Frontiers in Education Conference (FIE)*, pages 1–5, 2023. doi: 10.1109/FIE58773.2023.10343457.

Uwe Maier and Christian Klotz. Personalized feedback in digital learning environments: Classification framework and literature review. *Computers and Education: Artificial Intelligence*, 3:100080, 2022. ISSN 2666-920X. doi: https://doi.org/10.1016/j.caeai.2022.100080. URL https://www.sciencedirect.com/science/article/pii/S2666920X22000352.

M. Pankiewicz and R. S. Baker. Large language models (gpt) for automating feedback on programming assignments. In J.-L. Shin, A. Kashihara, W. Chen, and H. Ogata, editors, *31st International Conference on Computers in Education Conference Proceedings, Volume I*, pages 68–77. Asia-Pacific Society for Computers in Education (APSCE), 2023.

Lianne Roest, Hieke Keuning, and Johan Jeuring. Next-step hint generation for introductory programming using large language models. In *Proceedings of the 26th Australasian Computing Education Conference*, ACE '24, page 144–153, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400716195. doi: 10.1145/3636243.3636259. URL https://doi.org/10.1145/3636243.3636259.

Priscylla Silva, Evandro Costa, and Joseana Régis de Araújo. An adaptive approach to provide feedback for students in programming problem solving. In Andre Coy, Yugo Hayashi, and Maiga Chang, editors, *Intelligent Tutoring Systems*, pages 14–23, Cham, 2019. Springer International Publishing. ISBN 978-3-030-22244-4.

Gladys Tyen, Hassan Mansoor, Victor Carbune, Peter Chen, and Tony Mak. LLMs cannot find reasoning errors, but can correct them given the error location. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar, editors, *Findings of the Association for Computational Linguistics: ACL 2024*, pages 13894–13908, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.findings-acl.826. URL https://aclanthology.org/2024.findings-acl.826.

Shijie Xia, Xuefeng Li, Yixin Liu, Tongshuang Wu, and Pengfei Liu. Evaluating mathematical reasoning beyond accuracy. *arXiv preprint arXiv:2404.05692*, 2024.