Applying Large Language Models to Enhance the Assessment of Java Programming Assignments

Skyler Grandel
skyler.h.grandel@vanderbilt.edu
Vanderbilt University
Nashville, Tennessee, USA

Douglas C. Schmidt
dcschmidt@wm.edu
William & Mary
Williamsburg, Virginia, USA

Kevin Leach
kevin.leach@vanderbilt.edu
Vanderbilt University
Nashville, Tennessee, USA

Abstract

The assessment of programming assignments in computer science (CS) education traditionally relies on manual grading, which strives to provide comprehensive feedback on correctness, style, efficiency, and other software quality attributes. As class sizes increase, however, it is hard to provide detailed feedback consistently, especially when multiple assessors are required to handle a larger number of assignment submissions. Large Language Models (LLMs), such as ChatGPT, Claude, and Gemini, offer a promising alternative to help automate this assessment process in a consistent, scalable, and fair manner.

This paper explores the efficacy of ChatGPT-4 and other popular LLMs in automating programming assignment assessment. We conduct a series of studies within multiple Java-based CS courses at Vanderbilt University, comparing LLM-generated assessments to those produced by human graders. The analysis focuses on key metrics, such as accuracy, precision, recall, efficiency, and consistency, to identify programming mistakes based on predefined rubrics. Our findings demonstrate that LLMs improve grading objectivity and efficiency with appropriate prompt engineering and feature selection, serving as a valuable complementary tool to human graders in undergraduate and graduate CS education.

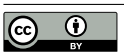
CCS Concepts

• **Software and its engineering** → *Software maintenance tools*; • **Applied computing** → **Computer-assisted instruction**.

Keywords

ChatGPT, Education, Generative AI, Large Language Models, Prompt Engineering, Automated Grading

ACM Reference Format:
Skyler Grandel, Douglas C. Schmidt, and Kevin Leach. 2025. Applying Large Language Models to Enhance the Assessment of Java Programming Assignments. In *33rd ACM International Conference on the Foundations of Software Engineering (FSE Companion '25), June 23–28, 2025, Trondheim, Norway*. ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3696630.3727236



FSE Companion '25, Trondheim, Norway

ACM ISBN 979-8-4007-1276-0/2025/06
https://doi.org/10.1145/3696630.3727236

1 Introduction

Motivating the need for more effective and scalable CS program assessment tools. The assessment of programming assignments in CS education traditionally demands substantial time and effort from instructors and graders. As CS class sizes continue to grow, this process becomes increasingly hard due to the risk of human error and subjectivity [29]. These issues are exacerbated when multiple graders are involved, leading to inconsistencies known as the *inter-rater reliability problem* [10, 29]. To address these challenges—and to improve the efficiency and objectivity of programming assessment—this paper investigates the potential of LLMs to automate and enhance the evaluation of student programs.

Conversational LLMs such as ChatGPT-4 [26] have demonstrated promising capabilities in diverse domains, including code generation and analysis [42]. These LLMs are particularly valuable when human expertise and AI tools can collaborate to address software-related challenges more efficiently and reliably [8, 41]. Such advances are increasingly applicable to educational contexts, particularly in fields where automated or assisted analysis of textual and programming content is beneficial [27, 34, 36].

To manage the challenges posed by large-scale classes, automated grading tools are commonly employed to assess various aspects of programming assignments. These tools often behave similarly to unit and integration test suites, focusing on functional correctness [7, 9, 29]. However, conventional auto-graders are inherently limited in their ability to evaluate other critical software quality attributes, such as coding style, efficiency, and broader software engineering principles like modularity, readability, and maintainability. Moreover, their reliance on strict structures may constrain student creativity, forcing submissions into overly rigid frameworks and pre-provided code "skeletons."

To complement functional assessments, "linter" tools are often used to enforce style guides and encourage the application of software engineering best practices [16]. However, conventional linters have notable limitations. For example, they do not holistically assess documentation quality, code readability, or stylistic coherence.

In contrast, LLMs offer a more versatile and qualitative approach to programming assessment, capable of evaluating functionality, coding style, efficiency, and other quality attributes in an automated and scalable manner. More broadly, the integration of generative AI tools into CS education has the potential to revolutionize pedagogical practices. For example, LLMs can provide personalized, adaptive feedback that goes beyond the capabilities of conventional auto-graders and test suites, fostering enhanced learning experiences for students [3, 7, 9, 29].

Solution approach → The GreAIter LLM-based auto-grading tool. To evaluate the effectiveness of generative AI at helping human graders locate faults and generate accurate/meaningful feedback for students in CS courses, we developed an LLM-based auto-grading tool known as "GreAIter." GreAIter converses effectively with ChatGPT-4 and other LLMs via *prompts* [42], which are natural language instructions provided to an LLM that customize it and/or enhance/refine its capabilities, thereby influencing its behavior. In turn, these prompts are guided by *prompt patterns* [41], which are a structured means of programming LLMs guided by experience [40, 44] with applying LLMs effectively.

This paper presents the results of case studies that applied GreAIter in the following CS courses:

- **Parallel Functional Programming** (https://www.dre.vanderbilt.edu/~schmidt/cs253, which focuses on modern Java parallel streams and reactive programming assignments in Java,
- **Scalable Microservices** (https://www.dre.vanderbilt.edu/~schmidt/cs891), which focuses on modern Java reactive concurrency assignments in Java and the Spring WebMVC and WebFlux frameworks, and
- **Programming and Problem Solving** (https://as.vanderbilt.edu/advising/caspar/academics/computer_science.php), which is an introductory CS1 [14] course that uses Java.

These case studies explore applying GreAIter to assess and grade programming assignments via a semi-automated grading methodology using ChatGPT-4 and other LLMs that supplement and enhance conventional auto-graders and traditional manual grading and code analysis.

GreAIter defines rubrics via JSON with specific grading criteria and communicates this method to LLMs. For each criterion contained in the rubric, GreAIter instructs the LLM to

(1) Output the code from each student submission that is relevant to that criterion along with a grade of "correct" or "incorrect" and then
(2) Compile a summary of all mistakes made by each student and outputs suggested feedback for students based on their submission and the mistakes therein.

The results of this assessment are reviewed by human graders to produce the final grade, thereby yielding an efficient, accurate, and fair grade by collaborating between humans and GreAIter.

To evaluate our approach, we analyzed GreAIter's performance in assessing programming assignments and compared its results with human graders. Specifically, we measured GreAIter's accuracy and efficiency in assessing student submissions using predefined rubrics. To identify limitations, we examined GreAIter's false positive and negative rates using precision and recall metrics. This analysis pinpointed GreAIter's shortcomings and evaluated its potential for automating the workflow of grading programming assignments.

We also investigated GreAIter's grading effort reductions and identified student mistakes it detected that human graders overlooked initially. This analysis highlighted the improvements enabled by AI-assisted grading and the common pedagogical oversights made by human graders in code evaluation. To ensure our approach generalized, we extended our evaluation to multiple LLMs to determine if GreAIter's effectiveness was LLM-specific.

This paper makes the following contributions to the field of AI-assisted programming assignment assessment:

- Empirical evidence demonstrating the utility of LLMs as tools for assisting in the assessment of programming assignments across both introductory and advanced computer science topics.
- Insights into the accuracy and effectiveness of LLMs in education, laying the groundwork for broader adoption of AI-assisted grading and further advances in this field.

This work builds upon our prior workshop publication [13], extending its scope and rigor in several key aspects:

- We generalize our original methodology to encompass a broader range of Java-based courses and LLMs, thereby enhancing the applicability and robustness of our findings.
- We include a detailed evaluation in a CS1 course, a foundational component of computer science education [14], to (1) demonstrate the efficacy of our approach in a critical and widely relevant context and (2) provide new insights into common human grading errors in CS1 and show how GreAIter mitigates these issues.
- We refine our assessment of GreAIter's intra-model consistency and repeatability, offering a more comprehensive analysis of its reliability.
- We expand the discussion and interpretation of our results, providing deeper insights and more robust conclusions regarding the role of AI-assisted grading in CS education.

Paper organization. The remainder of this paper is organized as follows: Section 2 describes our methodology, encompassing the design of our GreAIter auto-grading tool and the prompting strategies used to achieve our results; Section 3 explains the experiment we designed to assess the performance of our methodology in grading programming assignments in our three courses; Section 4 evaluates the results of our GreAIter grading tool using ChatGPT-4 and other popular LLMs; Section 5 explores the limitations and threats to the validity of our work; Section 6 compares our research with related work on AI-assisted programming assignment evaluation; and Section 7 presents lessons learned from our study and outlines future work.

2 Study Methodology

GreAIter is an LLM-based auto-grading tool that leverages advanced prompt engineering techniques and whose operation is supervised by human graders to ensure reliability and accuracy. This section describes the methodology underlying GreAIter and outlines the ethical considerations employed in this study. We adopt the methodology from our earlier workshop publication [13] and gather additional data following this methodology to compare directly with data collected earlier (Section 3 explains the additional courses, LLMs, and metrics we consider).

2.1 Overview of GreAIter and our AI-assisted Grading Process

Figure 1 depicts the steps involved in the GreAIter grading process, which begins with a student submission (1) and the evaluation rubric being input (2) into an LLM. This LLM then conducts an intermediate analysis (3) consisting of a detailed evaluation for each criterion in the rubric. The LLM then summarizes its assessment

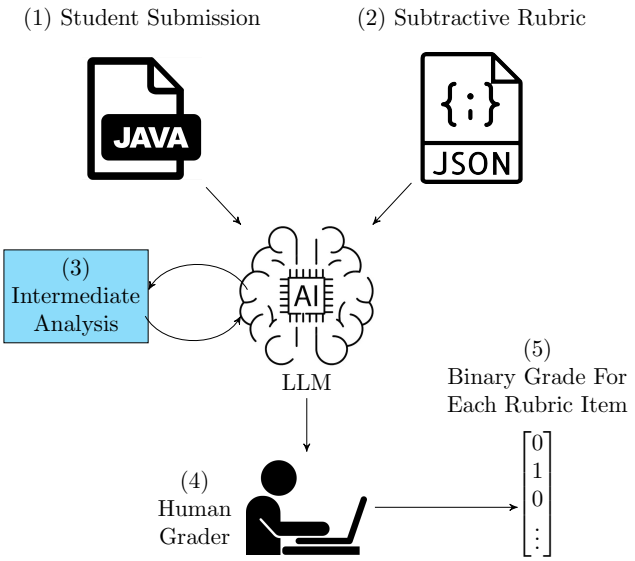


Figure 1: GreAIter's AI-Assisted Grading Process.

and a human grader reviews the output (4), verifying and adjusting the LLM's evaluations as needed. The final output from the human grader (5) consists of a binary grade (*i.e.*, pass/fail) for each criterion in the rubric, indicating the performance of the student's programming assignment submission against each criterion. This binary grade format was used for ease of calculating our evaluation metrics. In practice, however, the human grader does not need to be limited by this output structure and would be free to interpret the LLM's grade and student feedback in the way that best fits the course.

GreAIter bridges generative AI and education by helping instructors use LLMs to assess student work efficiently and objectively, aligning its outputs with human grading via rubric-based automation. It leverages *rubric-based evaluation* [17], where each criterion is clearly defined via a structured format. This format provides an LLM with the parameters needed to assess student submissions and ensure consistency across multiple evaluations, helping alleviate the inter-rater reliability problem [10].

Although GreAIter contains no features specific to a given programming assignment, we recommend certain steps when integrating it into a CS course.[1] For example, a reliable and structured rubric is needed for effective results. ChatGPT-4 has been shown [45] as an adequate prompt engineer proxy, so we leveraged it to generate rubrics used by GreAIter to prompt the LLMs.[2] In particular, ChatGPT-4 can generate a usable rubric given a (1) list of potential mistakes, (2) an "answer key" (*i.e.*, the desired Java assignment solution), and (3) the JSON shown in Figure 2. We used this approach to generate rubrics for our experiments, while manually verifying each rubric criterion ChatGPT-4 outputs to ensure the quality of its description, as well as correct and incorrect examples. The rubrics given to GreAIter were produced by converting the original course rubrics and answer keys to the JSON shown above.

[1]GreAIter is available to instructors of CS programming courses upon request. See Section 8

[2]We use ChatGPT-4 to refine our prompts, though other LLMs can perform these tasks.

```
{"[Filename]": {
    "criteria": [
      {
        "title": "[Title]",
        "description": "[In depth description
            of the criterion]",

        "correct_example":
        "[Correct example in your assignment's
            programming language]",

        "incorrect_example":
        "[Incorrect example in your assignment's
            programming language]"
      },
      .
      .
      .
    ]
}}
```

Figure 2: Example JSON that GreAIter Provides as a Rubric to the LLM.

Initial tests showed ChatGPT-4 sometimes generated flawed code for incorrect examples in rubric criteria that didn't match the types of mistakes students actually made. For example, we instructed ChatGPT-4 to ensure students use Java method references rather than lambda expressions, so a good solution might be "`.map(this::aMethod)`," whereas ChatGPT-4 would generate "`.map(item -> { })`" instead of "`map(item -> aMethod(item))`," which is semantically equivalent to the correct answer. We therefore provided examples of incorrect code via prompts to generate these rubrics and verified incorrect examples to ensure they exhibited realistic mistaken behaviors.

2.2 Prompt Engineering and Human-AI Collaboration

While GreAIter can operate fully autonomously, we applied a semi-autonomous method due to limitations with conventional LLMs [6, 40–42]. Despite their advanced capabilities, LLMs can generate errors (referred to as "hallucinations"), where they confidently assert inaccurate or nonsensical information [6]. This tendency is problematic for educational assessments, where the stakes of incorrect evaluations are high since they may impact a student's learning trajectory and academic record.

Given a programming assignment and rubric, GreAIter generated feedback for human graders to review. As a final sanity check, human graders then manually checked the relevant segment(s) of student code identified by GreAIter to verify that issues it flagged were indeed mistakes (rather than false positives). In practice, human graders inspect and score each student appropriately and review GreAIter's feedback before returning results to students via a FERPA-compliant course website or classroom code repository.

Integrating a human-in-the-loop introduced a crucial verification step. Human graders review the AI-generated assessments, ensuring the final output's reliability. This safeguard reinforces the educational value of the grading process. Human grader oversight ensures that feedback is pedagogically appropriate and contextually relevant to student learning needs.

Our semi-automated approach also aligns with ethical guidelines [15], promoting responsible AI use by mitigating risks associated with unverified autonomous AI operation in high-stakes

application domains, such as primary, secondary, and higher education. Our approach respects the sophistication of the AI while prudently managing its limitations and balancing the efficiency of automation without foregoing the expertise of humans. The result is a hybrid model that aims for high-quality, scalable assessment mechanisms that educators and students alike can rely upon.

GreAIter's functionality stems from *prompt engineering* [41], which is the intentional design of prompts that guide LLMs in performing their tasks. Prompts for GreAIter are crafted carefully to elicit specific behaviors from the LLM, enabling it to understand and apply grading rubrics accurately. We encoded the rubrics using JSON (as shown in Figure 2) since LLMs handle this format accurately. Each rubric was defined as a JSON array, where each element contained an object representing a rubric criterion. Each rubric criterion contained entries for the criterion's title, description, and a correct and incorrect example.

The following prompt instructed the LLM how to use this rubric:

> You are a grader for the [course name] course taught in Java. I will give you a JSON rubric and student Java code. For each item in the rubric, you will first output the method in the student's code that is relevant to that item and then you will output a score of "correct" or "incorrect". Alternative answers to the correct code are permissible if they have the same functionality and do not apply poor Java style conventions.

The rubric and the student's code followed this prompt. The next prompt instructed the LLM to compile a comprehensive summary of errors or misalignments with rubric expectations, along with suggested feedback for the student based on their specific mistakes.

This sequence of prompts forced the LLM to consider each individual criterion in the rubric. A *chain-of-thought*[3] prompting strategy was then employed by instructing the LLM to output the relevant code before making a judgment about its correctness. This process helped minimize LLMs' tendencies to hallucinate, skip rubric criteria, and/or consider irrelevant parts of student code.

During this study, we found it was crucial to include both correct and incorrect examples in the rubric since it enabled *few-shot learning*[23] that trains an LLM from only a few examples. Few-shot learning typically performs better than for a 0-shot approach [39] by providing context, clarifing edge cases, and alleviating potentially ambiguous instructions. Likewise, these examples aided the LLM in providing specific feedback to students by comparing their solutions with the desired solution.

2.3 Assessment Process and Ethical Considerations

GreAIter's assessment process began with an LLM receiving each student's code and the associated rubric via our prompts, as shown by step (1) in Figure 1. The LLM then systematically evaluated the code, criterion-by-criterion, referencing specific code segments from the student's submission as evidence for its assessments. Our prompts were designed to ensure that the LLM's evaluation was not merely keyword-based but contextually rooted in the logic and syntax required by the rubric.

[3]Chain-of-thought prompting [40] instructs an LLM to explain its "thought process" before giving an answer to improve answer quality.

Our experiments obtained IRB approval for use of student data, which was de-identified prior to any contact with any LLM. In addition, we used a university-hosted private Azure instance of each LLM to minimize the risks of data leaks. Assignments were passed to this private instance via an API similar to the OpenAI API and the LLM was instructed to grade each criterion in the given rubric as it applied to each student's code.

After GreAIter completed assessing each criterion, it aggregated individual assessments into a final summary. This summary conveyed areas where the student excelled and areas that required further improvement. This summary also provided a foundational tool for human graders to either validate the results of GreAIter's grading process or provide additional insights where necessary.

To ensure assessment integrity, the outputs generated by GreAIter were cross-examined by human graders. This hybrid approach fine-tuned the assessment process and established a comprehensive feedback system that benefited student learning experiences [5, 31]. Our goal was to harness the computational precision and scalability of LLMs while retaining the nuanced judgment of humans, striving for an equilibrium that augmented the grading process within our university and similar educational environments.

3 Experiment Design and Evaluation

To evaluate our methodology described in Section 2, we designed an experiment to empirically determine how well GreAIter performed in its assessment of student assignments. This experiment evaluated the performance of our GreAIter automated code assessor against human graders in terms of *accuracy*, *efficiency*, and *objectivity*. The experimental setup described in this section measured the efficacy of GreAIter by comparing its assessment outcomes to those of experienced human graders.

Assignments and student submissions for this experiment were obtained from the following three courses at our institution, all taught in Java, but varying in assignment size and topics covered:

- The **Parallel Functional Programming** course consisted of 26 undergraduate and graduate students in the fall semester of 2023 and had four assignments covering functional paradigms in Java, as well as topics in parallel and asynchronous programming.
- The **Scalable Microservices** course consisted of 44 undergraduate and graduate students in the spring semester of 2024 and had two assignments covering efficient networking, database querying, and data processing in microservice environments using Java's Spring frameworks.
- The **Programming and Problem Solving** course consisted of 168 undergraduate students in the spring of 2024. We evaluate nine assignments covering topics like Java structures and syntax, handling I/O, and basic problem solving.

For each assignment, every student submission was graded using our semi-automated approach and student mistakes identified by GreAIter were recorded. We initially intended to use final student grades on the assignments as the "ground-truth" human-graded benchmark and compare them to GreAIter's output for the same student assignments. However, since each assignment was graded using our semi-automated approach, we encountered instances where human graders missed student mistakes. We thus recorded these human grader mistakes and used the corrected scores as

ground-truth for comparison. Our experiment considered the following three research questions:

RQ1: Performance. Can GreAIter perform correctly by identifying mistakes in student program submissions?

RQ2: Efficiency. What is the reduction in the amount of manual grading that must be done when using GreAIter compared against traditional manual grading?

RQ3: Consistency. How consistent is GreAIter across multiple grading attempts of the same programming assignments?

Section 4 discusses our recommendations for integrating GreAIter as a semi-automated grader in CS classes. For this experiment, however, we assessed GreAIter's performance in isolation to provide evidence for our recommendation, except where we considered the time saved by our methodology compared with conventional manual grading. While GreAIter could fully automate grading, we designed the experiments described below to identify GreAIter's failure modes, evaluate its efficacy, and determine how its output can be verified in an AI-assisted grader process. Unless otherwise specified, these results applied ChatGPT-4 as our subject LLM (experiments involving other LLMs can be found in Section 3.3).

3.1 RQ1: Performance

Building upon the experimental design described above, we used three performance metrics to evaluate GreAIter rigorously.

3.1.1 Accuracy. GreAIter's *accuracy* was quantified as the percentage of student mistakes (*i.e.*, missed rubric criteria) correctly identified by GreAIter in alignment with the ground-truth grades. High accuracy results helped validate GreAIter as a reliable evaluator of code quality and correctness. In turn, this motivated its integration into the grading process to reduce the grading load on instructors and graders, while maintaining high assessment standards.

3.1.2 Precision. GreAIter's *precision* was quantified as the extent to which it *incorrectly* marked a correct code segment as erroneous. High precision indicated GreAIter rarely marked correct code as erroneous, preventing undue penalties on students and minimizing human oversight. High precision also indicated GreAIter's meticulousness, ensuring its feedback was constructive and based on actual student errors, thus building student trust in its assessment. While poor precision impacts GreAIter's ability to operate autonomously in isolation, this problem can be mitigated with more human grader intervention in the overall GreAIter assessment process.

3.1.3 Recall. GreAIter's *recall* was quantified as its ability to identify all incorrect code present. A high recall rate indicated GreAIter could effectively detect most—if not all—errors in student submissions, which is critical because identifying mistakes demonstrates its ability to assist human graders. In contrast, a low recall rate would require so much human oversight that GreAIter would not accelerate assessments significantly as class sizes increase. GreAIter's ability to provide comprehensive feedback for educational purposes would be enhanced if it exhibited high recall, *i.e.*, if it consistently identified mistakes that human graders could be verified quickly.

3.1.4 Summary of Performance Metrics. A summary of results for GreAIter's performance metrics is shown in Table 1, which depicts the results of our AI grading methodology versus human graders, broken down by course and assignment. We use abbreviations for course titles: **PFP** stands for Parallel Functional Programming, **SM** stands for Scalable Microservices, and **P&PS** stands for Programming and Problem Solving (the introductory CS1 course). The aggregated results are displayed in the final bolded row.

Table 1: GreAIter Performance Metrics.
Accuracy, Precision, and Recall are depicted as percentages.

Assign.	TP	FP	TN	FN	Sum	Acc.	Prec.	Rec.
PFP1	15	32	811	0	858	96.27	31.91	100
PFP2	29	16	345	0	390	95.90	64.44	100
PFP3	4	0	698	0	702	100	100	100
PFP4	6	4	510	0	520	99.23	60.00	100
SM1	103	29	1,144	0	1,276	97.73	78.03	100
SM2	10	13	391	0	414	96.86	43.48	100
P&PS1	48	7	1,289	0	1,344	99.48	87.27	100
P&PS2	63	9	1,248	0	1,320	99.32	87.50	100
P&PS3	99	11	2,695	0	2,805	99.61	90.00	100
P&PS4	59	35	2,530	0	2,624	98.67	62.77	100
P&PS5	126	24	1,969	0	2,119	98.87	84.00	100
P&PS6	145	43	2,947	0	3,135	98.63	77.13	100
P&PS7	116	94	2,190	0	2,400	96.07	55.24	100
P&PS8	56	29	1,056	0	1,141	97.46	65.88	100
P&PS9	93	46	988	0	1,127	95.92	66.91	100
Total	**972**	**392**	**20,811**	**0**	**22,175**	**98.23**	**71.26**	**100**

We benchmarked GreAIter's performance against the corrected original grades for each course, yielding the following results:

- **True Positives (TP): 4.38%** - The percentage of instances where GreAIter correctly identified errors that were also recognized by the TA grader.
- **False Positives (FP): 1.77%** - The percentage of instances where GreAIter marked correct code segments as erroneous.
- **True Negatives (TN): 93.85%** - The percentage of instances where GreAIter correctly identified correct code segments, aligning with the TA grader's assessments.
- **False Negatives (FN): 0%** - The percentage of instances where GreAIter marked erroneous code segments as correct.
- **Accuracy (Acc.): 98.23%** - The proportion of correct assessments made for all grading decisions.

Minimizing false positives and false negatives is a crucial aspect of ensuring fair and constructive feedback. The precision and recall of GreAIter reflect its ability in these regards, as follows:

- **Precision (Prec.): 71.26%** - GreAIter's ability to correctly identify student mistakes without over-penalizing correct aspects of their submissions.
- **Recall (Rec.): 100%** - The comprehensiveness of GreAIter in detecting errors present in the student submissions.

These results show that GreAIter can enable human graders of programming assignments to catch additional mistakes they might miss otherwise. In particular, graders in the CS1 course (Programming and Problem Solving) often missed mistakes, likely because they were less experienced undergraduate students, who may also be overworked considering the high number of students in this class. While testing GreAIter to obtain our performance metrics, we found 519 mistakes that were missed by the original human

graders. These mistakes are broken down by topic in Table 2 to inform pedagogical research about this problem.

Table 2: Mistakes Missed by Human Graders in P&PS (CS1)

Type	Comments	Constants	Readability
Count	355	24	54
Type	**Variable Naming**	**Modularity**	**Functionality**
Count	31	30	25

Specifically, these topics refer to poorly descriptive or missing comments, misuse of constants, misuse of whitespace or other readability concerns, poorly descriptive variable names, poor modularity among methods (*i.e.*, placing all code in the main method or creating several nearly identical methods without appropriate code reuse), and general mistakes in code functionality. While code functionality mistakes can be resolved via unit tests, these mistakes constitute edge cases that testing did not account for (*e.g.*, one submission used `System.exit()` to exit a loop, which halts the entire program).

These types of mistakes show that the original human graders for the P&PS course achieved an accuracy of 97.12%, which is 1.11% lower than GreAIter without human oversight. Since GreAIter caught the mistakes made by human graders in the P&PS course, we conclude that human graders using GreAIter in CS1 courses will be more accurate, as well as expending less effort when compared to strictly manual grading.

3.1.5 Analysis of Performance Metric Results. The results of applying our semi-automated GreAIter tool yielded several observations. Our high overall accuracy rate indicates a strong foundational reliability of ChatGPT-4 in evaluating Java programming assignments. Interestingly, the seemingly low true positive rate of 4.38% indicates we had somewhat skewed data, with considerably more true negatives than true positives. This result is not unexpected, however, since student grades on programming assignments in these courses tended to average over 90%. Nevertheless, it does render our true positive rate somewhat meaningless, so we focus on accuracy, precision, and recall instead.

A precision of 71.26% indicates a tendency towards false positives, where GreAIter incorrectly marks valid code as erroneous. Although GreAIter is thorough, it may be overly critical or prone to hallucinations, *i.e.*, perceiving errors that are not there. More optimistically, GreAIter exhibited perfect recall in this experiment, as it never missed a mistake made by a student.

Overall, these results suggest that while GreAIter shows promise in terms of high accuracy and recall, it should be managed carefully due to its propensity for false positives. The strong recall indicates that GreAIter can serve as an effective initial filter in identifying potential errors in student submissions, potentially even improving upon human grader accuracy. However, GreAIter's precision underscores the necessity of human oversight to confirm LLM findings and to provide the final judgment on the student's work, which substantiates our focus on a semi-automated grading approach.

3.2 RQ2: Efficiency

To address the second research question, we focused on evaluating the efficiency of GreAIter's LLM-based grading system compared to traditional manual grading methods. We defined efficiency as (1) the time investment required for grading, (2) the number of rubric criteria a grader must assess, and (3) the volume of code that must be reviewed for each submission. We summarize the results of these experiments in Table 3, which shows the reduction in time to grade, as well as the reduction in rubric criteria and source lines of code to check. We show the reductions for each course individually and the broader results since these three different courses cover a range of topics and project sizes.

Table 3: Time and Effort Reductions Achieved by GreAIter

Class	Time	Rubric Criteria	Code Volume (SLOC)
PFP	82.59%	95.71%	96.37%
SM	78.84%	90.83%	96.55%
P&PS	52.36%	93.48%	91.75%
Total	**75.82%**	**93.85%**	**93.21%**

3.2.1 Time Investment. Based on our experience applying GreAIter throughout our experiments, we found that its semi-automated grading process was notably faster than manual grading. In particular, we observed that GreAIter's runtime averaged 51 seconds per student submission due largely to the request latency of our API. However, GreAIter could simultaneously assess *all* submissions for a given assignment, so it could run as a background process while human grader(s) reviewed the results. The runtime of GreAIter thus had a negligible effect on overall grading efficiency.

The time needed to grade each submission is a critical measure of efficiency. We tracked the duration it took our semi-automated GreAIter method to complete the grading process for all student submissions in a given assignment and calculated the time to grade a single student submission averaged over all assignments. Our observations indicated that GreAIter *substantially* reduced the time required per submission, *i.e.*, the average time taken by a human grader using GreAIter for a single student submission was roughly 87 seconds.

For comparison, we measured the time needed to grade 10 student submissions manually for each assignment to calculate the average time needed to grade each assignment without AI assistance. This task was performed independently by two researchers without overlapping assignments, so 20 submissions were graded for each assignment to mitigate concerns regarding a single subjective grader's experience and skill. These assignments were chosen via random stratified sampling based on the students' final corrected grades on the assignment to ensure that the subset was representative of the overall set.

In contrast to the GreAIter-based approach, the manual graders spent an average of ~360 seconds (*i.e.*, ~6 minutes) per submission. GreAIter thus reduced overall grading time by ~**75.82%**, highlighting its ability to accelerate grading efficiency in educational settings. This increased efficiency is more pronounced in much larger CS classes requiring multiple human graders and also addresses the inter-rater reliability problems arising with multiple human graders.

3.2.2 Number of Rubric Criteria. Another dimension of efficiency we investigated was the number of rubric criteria a grader must

check. In the traditional manual method, each grader must assess each criterion for every submission. In contrast, GreAIter's ability to identify correct code segments quickly reduced the number of criteria requiring detailed human review.

We quantified the average number of rubric criteria the human grader had to assess in-depth for each submission compared to those GreAIter flagged for further review. Our findings showed that a human grader using GreAIter only needed to review roughly 0.8 rubric criteria per student submission on average, which was substantially less than the average of 13 criteria for the human grader alone. Overall, this result constituted a substantial decrease of **93.85%** due to GreAIter's high recall, low false negatives, and high true negatives.

3.2.3 Volume of Code. Finally, we evaluated efficiency in terms of the total number of lines of code a human grader needed to check to grade a submission. GreAIter's capability to target relevant code segments precisely for each rubric criterion reduced the overall volume of code that needs in-depth review, due to its ability to locate the relevant Java method to each rubric criterion within student code. GreAIter thus only needed to check Java methods that an LLM highlighted as relevant to a rubric criterion that a student missed.

We conclude that a human grader without AI assistance must review nearly every line of code in a submission. In contrast, a human grader using GreAIter only needs to check the Java method identified as having a defect. We therefore compared the average number of lines of code requiring review (under this assumption) per submission by a grader with and without GreAIter.

In this experiment we counted Source Lines of Code (SLOC) for comparison, *i.e.*, lines of code not including blank lines, comments, and imports [25].[4] Overall, GreAIter required a grader to review an average of 7.2 lines of code per student submission. In contrast, a grader without AI assistance needed to review an average of 106 lines of code, constituting a **93.21%** decrease in volume for GreAIter.

Our experiment results indicate a substantial improvement in grading efficiency when using ChatGPT-4 as the LLM for GreAIter. Grading workload dropped significantly thanks to GreAIter's faster review times, fewer rubric criteria needing deep evaluation, and less code to inspect per submission. Moreover, this efficiency increase does not degrade grading quality since GreAIter still adhered to our rubric grading standards. By freeing up time and effort, moreover, human graders using GreAIter focused more on providing quality feedback, engaging in interactive teaching, and developing better course content, thereby enriching the overall student educational experience.

3.3 RQ3: Consistency

To assess GreAIter's reliability, we implemented a repeatability test that measured its consistency over time. We defined consistency as GreAIter's ability to produce the same results when presented with the same inputs under similar conditions. This ability is vital for its deployment in educational settings and ensures that GreAIter grades fairly between submissions.

[4]We exclude imports because the imports for these assignments were given to students, and were thus not considered while grading.

3.3.1 Intraclass Correlation Coefficient Metric. The primary metric for success in our repeatability test is the *Intraclass Correlation Coefficient (ICC)* [20]. Specifically, we use McGraw and Wong's two-way random effects, consistency, and single rater formulation of ICC [22]. This formulation measures the consistency between ratings based on the grade produced by a single "rater" (*i.e.*, a single instance of ChatGPT-4). A high ICC indicates that GreAIter is stable and reliable in both its grading and feedback and exhibits minimal inter-rater reliability bias, which is essential for any (semi-)automated grading tool used in academia.

To conduct this experiment, we sampled 20 representative student submissions via random stratified sampling and used GreAIter to produce AI-generated grades for each of the student submissions six times (including the initial round of grading used to produce the results in RQ1). We then calculated the ICC between the six grading results.

The ICC we observed was **0.80**, with a 95% confidence interval of [0.68, 0.9] and a p-value less than 0.001, indicating "good reliability" [20]. However, all identified disagreements were false positives, *e.g.*, in one trial GreAIter hallucinated a problem while it did not hallucinate (or hallucinated a different problem) in the other trial. This finding highlights the stability of GreAIter and its minimization of grader bias.

This consistency metric indicates the repeatability of GreAIter's performance. Although its rate is not perfect, it is substantial and shows how GreAIter can reliably reproduce its grading decisions across multiple iterations. The inconsistencies we observed arose from false positives, reinforcing our earlier observation that GreAIter tends to over-diagnosis errors. GreAIter's inconsistencies are thus consistent with our previous findings that underscore the necessity of human oversight to confirm GreAIter's findings and provide the final judgment on student programming submissions.

3.3.2 GreAIter's Consistency Across Multiple LLMs. We investigated the consistency of GreAIter across multiple LLMs to quantify the impact of choosing a particular LLM. We tested four LLMs on a representative subset of 40 student submissions, sampled through random stratified sampling in the same way as the intra-LLM consistency and timing tests. We used the same prompts for each LLM and graded each submission using our methodology, varying only the LLM used. The results using each LLM are shown in Table 4.

Table 4: A Comparison of LLMs Using GreAIter.

Model	TP	FP	TN	FN	Acc.	Prec.	Rec.
GPT-4	**30**	22	468	**0**	0.9577	0.5769	**1.0000**
GPT-4o	**30**	11	479	**0**	**0.9788**	**0.7317**	**1.0000**
Claude	25	**10**	**480**	5	0.9712	0.7143	0.8333
Mistral	25	26	464	5	0.9404	0.4902	0.8333

This table shows the True Positives (TP), False Positives (FP), True Negatives (TN), False Negatives (FN), Accuracy (Acc.), Precision (Prec.), and Recall (Rec.) for various LLMs, including GPT-4, GPT-4o, Claude Opus [1], and Mistral Large 2 [24]. The best performance in each metric is highlighted in bold. These results show that GPT-4o performs best based on our metrics. In contrast, Claude Opus and Mistral Large 2 do not achieve perfect recall, likely because our prompts were not fine-tuned for these LLMs.

Table 4's results show the LLM's impact on GreAIter's grading accuracy. Our evaluations depict how LLMs exhibit varying levels of accuracy, precision, and recall, which highlights the need to select an LLM carefully to ensure GreAIter's reliability. Moreover, while baseline prompts were used uniformly across LLMs, optimizing these prompts for specific LLMs could improve performance consistency, especially for Claude Opus and Mistral Large 2 that exhibited lower recall. These results show how choosing an LLM and fine-tuning prompts are critical factors in maximizing the effectiveness of GreAIter's semi-automated grading process.

4 Analysis of Results

This section analyzes the results of our semi-automated GreAIter tool, focusing on the implications of its performance metrics, its potential role and integration within educational settings, and considerations for its future application. Table 5 summarizes our experimental results using the corrected grader outcomes as the ground truth, as discussed in Section 3.

Table 5: Summary of Results.

Performance	Accuracy 98.23%	Precision 71.26%	Recall 100%
Effort Reduction	Time 75.82%	Rubric Criteria 93.85%	Code Volume 93.21%
Consistency	Intraclass Correlation Coefficient 0.80		

Performance metrics and efficiency gains. GreAIter's high overall accuracy (98.23%) and recall (100%) indicate its utility as a tool for assessing programming assignments. Its effectiveness in correctly identifying correct code submissions significantly reduced grader workload by filtering out submissions that required no further review. However, its precision of 71.26% raises concerns regarding its number of false positives, which reflects the limitations of ChatGPT-4 that misinterpret complex instructions or code nuances, leading to the incorrect identification of errors.

This outcome led us to apply *semi-automated* grading and feedback by using GreAIter to find candidate mistakes in student code for later human grader review. GreAIter shows the relevant code for each mistake. Human graders can quickly validate candidates, which (1) accelerates grading and reduces human effort, as shown in Section 3.2 *and* (2) maintains human intervention to mitigate any student concerns of AI-based assessment accuracy and fairness.

According to our inter-LLM consistency study, however, the perfect recall achieved by GPT-4 and GPT-4o does not extend to other LLMs in the absence of prompt fine-tuning for those LLMs. In particular, Mistral Large 2 may not be well suited to this task with our current prompting, though Claude Opus's performance is comparable to GPT-4o (recall notwithstanding). While the training methodologies for the Claude and GPT models are opaque, we suspect they prefer longer and more detailed responses, but their means of achieving greater length are different.

In our experience thus far, GPT-4 prefers finding as many mistakes in the student's code as possible to achieve a longer output, often leading to harsher grading or hallucinated mistakes. In contrast, Claude Opus prefers lengthening its output by detailing the positive aspects of a student's code. In the specific cases of the Claude Opus's false negatives that we observe, we find that this model will give lengthy discussions, often in several paragraphs, describing ways the student was correct. We therefore conclude that while GPT-4 and GPT-4o are better suited for this task, Claude Opus might be preferred for other grading applications, such as generating feedback for student submissions.

Repeatability and semi-automation. The ICC of 0.80 suggests that while GreAIter achieves "good reliability," there are some variations in its grading across iterations. This variability can be problematic in CS courses, where consistency in assessing programming assignments is essential to fairness and grading process credibility. However, because disagreement between trials stems entirely from false positives, GreAIter can help focus human grader attention on reducing bias and improving grading efficiency by decreasing grading time by up to 75.82%.

Our results also show how GreAIter contributes to the grading process via initial assessments and identification of clear-cut cases of correct code. Its high accuracy in these instances enable instructors to focus on providing in-depth feedback where it is most needed, thereby enhancing student educational experience. Nevertheless, GreAIter's propensity for false positives necessitates a semi-automated approach where human graders perform a secondary review of its grading decisions. This approach leverages the strengths of GreAIter in rapidly processing and evaluating submissions while mitigating its weaknesses through human oversight.

Overall, our semi-autonomous "augmented intelligence" approach (*i.e.*, where GreAIter provides a first pass and humans verify) offers a balanced solution, combining the thoroughness and speed of LLMs like ChatGPT-4 with the discernment and expertise of human graders. This hybrid strategy helps streamline the grading process, reduce workload for instructors and graders, and maintain the integrity and fairness expected of academic evaluations.

5 Limitations and Threats to Validity

This section explores limitations with the GreAIter AI-assisted grader described in Section 2 and threats to validity of our experiments described in Section 3.

Generalizability across languages and paradigms. Although our study is extensive, it has limitations. In particular, we developed and evaluated GreAIter within three courses and one programming language (Java), which may limit our finding's generalizability. The topics covered by these classes and their use of Java may incur unique challenges and patterns not representative of other curricula, programming languages, or paradigms.

Subjectivity in ground truth and precision concerns. We compared GreAIter's performance with the corrected original graders from each course. This evaluation may introduce a degree of subjectivity into the ground-truth against which GreAIter's performance was measured. In particular, different graders may have different thresholds for correctness and error severity, potentially influencing the evaluation benchmarks. In addition, our corrections to these grades were made only in egregious cases, so the original grades and our corrections could have a degree of subjectivity to them.

For instance, our recall metrics may be affected if human graders overlook errors in student code. To address this, we only counted cases where GreAIter flagged a mistake that human graders missed,

i.e., false negatives. In such cases, GreAIter correctly identified a potential issue and it was up to human graders to decide whether it warranted a deduction. It's therefore reasonable to treat GreAIter as accurate when it flags a possible error. To strengthen our findings, however, it would be prudent to validate results across more graders and courses, and to compare human inter-grader reliability.

The false positives reported in GreAIter's outcomes reflect another limitation since false positives in code review tasks can lead to user frustration and tool abandonment [18, 28]. While GreAIter's recall was perfect (indicating it missed no errors) its precision was lower, suggesting it sometimes identified errors too zealously. While this over-detection erred on the side of caution, it triggered unnecessary manual reviews, diminishing GreAIter's efficiency gains.

Consistency and repeatability concerns. Another threat to study validity is our reliance on API latency, which may affect grading speed results. Likewise, our consistency measure assumes LLMs don't learn or adapt over time, which doesn't account for continuous enhancements to their underlying models. However, this metric simply verifies the consistency of grades within a single cohort for a single assignment, so base model updates should not vary within a single assignment.

Despite these issues, our study provides insights into the capabilities and limitations of using LLMs for assessing programming assignments. GreAIter's high accuracy and recall indicate its potential utility in CS courses. Likewise, its Intraclass Correlation Coefficient is promising, though imperfect. The careful design of our study, the systematic approach to data collection and analysis, and the critical evaluation of results all contribute to the robustness of our findings. These limitations provide a clear framework for understanding the context within which the findings are applicable.

6 Related Work

AI-assisted education is a rapidly evolving field that shapes our approach to automating programming assignment assessment using LLMs and prompt engineering patterns. This section places our work within the broader landscape of prompt engineering and AI-powered educational tools. Moreover, this paper is an extension of a previous workshop publication [13] with a much more extensive evaluation methodology and discussion, among other improvements summarized in Section 1.

Prompt engineering to improve LLM reasoning. Past studies explored how prompt engineering enhances LLM performance, ranging from simple prompt strategies [2] to more sophisticated methods [41, 44]. Wei et al. [40] introduced "chain-of-thought" prompting, a technique we incorporate in our work to improve the reasoning capabilities of LLMs. Building on this, Yao et al. [44] proposed a more advanced framework that integrates action plan generation and external resource lookup to refine LLM performance.

Similarly, White et al. [41] developed prompt patterns——which are analogous to software patterns——that structure and enhance LLM outputs. Their subsequent work demonstrated the effectiveness of these patterns in improving code quality specifically [42]. In parallel, research on common failure modes of LLMs [6] has provided insights that guide mitigations to improve robustness and reliability, which we consider in our study.

LLMs in educational assessment. The application of AI in education has been explored extensively in prior work. Of particular relevance is a study by Schneider et al. [34] on automated grading of short-answer questions using LLMs. Their findings highlight the importance of human oversight in achieving reliable outcomes, a key consideration in our own evaluation. This work extended an earlier study [33] that employed fine-tuned transformer models, which achieved superior accuracy compared to traditional automated methods. Broader discussions on the benefits and challenges of AI in education [3, 15, 30, 36, 43] further underscore the need for careful evaluation and integration of LLMs into pedagogical practices, which our study addresses.

We also align our work with recommendations for the responsible use of LLMs in education by van Dis et al. [38]. This work advocated for approaches that "embrace the benefits of AI" while retaining "human verification." Following these principles, we propose a semi-automated assessment methodology that leverages AI capabilities while maintaining human oversight to ensure accuracy and fairness.

Additional research highlights educational applications of LLMs, including the detection of AI-generated submissions [27], generation of programming exercises and explanations [32], and support for medical education [21]. These studies demonstrate the versatility of LLMs in enhancing pedagogy, providing a foundation for our exploration of LLMs in programming assignment assessment.

Limitations of traditional grading tools. Automated grading systems for have a long-standing tradition in CS education. Test suites and linters are widely used to evaluate functional correctness and enforce style guidelines [9, 12, 29]. While these tools are effective for functional assessment, they often lack the ability to evaluate qualitative aspects of code, such as documentation quality, holistic readability, and efficiency. Our work leverages LLMs to address these limitations, enabling a more comprehensive and flexible assessment methodology.

Our study advances the growing body of research in prompt engineering, AI-assisted education, and automated grading systems. By applying sophisticated prompt engineering techniques to LLMs, we introduce a novel methodology for assessing programming assignments. This approach not only improves grading efficiency and reduces bias but also highlights the broader potential of AI to enhance learning experiences in computer science education.

7 Concluding Remarks

This paper presented the results of our study that applied LLMs to create GreAIter. GreAIter is an AI-assisted tool that helps automate key portions of the grading process for programming assignments to better assess students' code and its alignment with software engineering best practices. Our findings codify both the potential and limitations of AI-assisted grading, as well as yielded the following lessons learned that inform our future work.

1. ChatGPT-4 can accurately identify correct code submissions. We demonstrate how GreAIter achieved a high accuracy rate (98.23%) and perfect recall. These results suggest that LLMs can play an important role in assisting with grading tasks, particularly in filtering out rubric criteria that are likely correct for a given submission, thereby reducing human grader workload. Ironically, when ChatGPT-4 did not accurately identify correct code submissions, we interacted with it to explain how we could craft future prompts to elicit more accurate results from it. The ability to engage in such

a "Socratic dialogue" with LLMs was quite refreshing compared with traditional means of refining queries with conventional static analysis tools.

2. The need for human oversight remains critical. GreAIter's precision of 71.26% (marked by many false positives) points to limitations with the current state of LLMs in understanding and evaluating complex programming tasks. Despite GreAIter's perfect recall, human oversight thus remains necessary due to LLM tendencies to over-flag student code segments as erroneous. By integrating insights from previous work with GreAIter, we extend the capabilities of LLMs in educational contexts and set the stage for future research in AI-assisted education. As with previous work [4, 8, 35, 41], the synergy of LLMs and human expertise demonstrated in this study showcases the potential of LLMs in enhancing educational methodologies and outcomes.

3. Due to ChatGPT-4's limitations, we stress the benefits of a semi-automated grading approach. Our study underscored the importance of human-AI collaboration, which is commonly known as "augmented intelligence" rather than "artificial intelligence." While GreAIter is powerful, it cannot yet replace human judgment in tasks requiring nuanced understanding. Our semi-automated approach—where ChatGPT-4 performed an initial assessment and humans provide final verification and feedback—strikes a balance that leverages the strengths of both. We found our approach reduced grading workloads, constituting a 93.21% decrease in code volume to review and a 75.82% decrease in grading time. We also found GreAIter yielded an ICC of 0.80, indicating that while LLMs can exhibit consistent, relatively unbiased tendencies, their performance can vary and should be checked for accuracy.

4. GreAIter can be integrated into actual classroom settings to improve grader efficiency and reliability. Through careful planning and systematic analysis of the accuracy, precision, recall, efficiency, and repeatability metrics covered in this paper, we showed how GreAIter improves traditional manual grading. We validated the feasibility of integrating GreAIter into actual classrooms to optimize the grading process.

While GreAIter provides a high degree of accuracy, its current limitations underscore the need for a semi-automated approach that combines the speed and consistency of LLMs with the critical thinking and expertise of human graders. As LLMs evolve, so too will strategies for integrating it more effectively to enhance quality and fairness of the grading process for CS courses. Our future work will thus investigate LLMs that have emerged since our experiments began, including ChatGPT o3 and o4, as well as DeepSeek, Grok, and Gemini. Following these experiments, we may need to reassess whether these LLMs can be "trusted" to perform more autonomously if they achieve near-perfect precision.

5. The promise of LLMs in education extends beyond grading efficiency, *i.e.*, LLMs can reshape how feedback is delivered, how learning is assessed, and how education is ultimately conducted. Our work, along with several others [11, 19, 37, 43], has shown the potential LLMs have to aid student learning. While LLMs are not yet mature enough to replace human graders, they provide an important resource to aid educators, particularly in disciplines like CS characterized by ever-growing class sizes.

As with previous research [41, 42], we found collaboration between human users and AI tools results in rapid and reliable software solutions, and we leveraged this collaboration for a more efficient and effective educational process. As LLMs become more sophisticated, we anticipate the need to refine these powerful tools to improve educational systems. Our results provide a foundation upon which future work can build, contributing to the evolving field of LLMs in CS education and the development of more reliable and efficient AI-assisted graders.

Looking ahead, the integration of LLMs into grading CS programming assignments requires considering the trade-offs between efficiency and accuracy. False positive rates must be reduced to make AI-augmented tools like GreAIter more autonomous and trustworthy.[5] Our future work will focus on fine-tuning LLMs on larger and more diverse datasets of code submissions and rubrics, potentially improving their understanding and reducing the rate of false positives. Many false positives result from the same rubric criteria, so investigations into prompting strategies for specific rubric criteria are also likely to improve precision.

While our case study is bound by these limitations for now, our methodology and the GreAIter's overall performance in several key metrics support the validity of our findings. By advancing the tools used to assess code quality, GreAIter thus lays the foundation for CS education to produce more skilled and productive software engineers.

8 Data Availability

All data and scripts, along with our implementation of GreAIter are available at https://osf.io/xy69w/?view_only=9a71d5fdec10447d8c85ac7e3da9b1fa. This repository includes comprehensive documentation to facilitate replication and extension of our research.

References

[1] Anthropic. 2024. *Claude Opus.* https://claude.ai/
[2] Simran Arora, Avanika Narayan, Mayee F Chen, Laurel Orr, Neel Guha, Kush Bhatia, Ines Chami, Frederic Sala, and Christopher Ré. 2022. Ask me anything: A simple strategy for prompting language models. *arXiv preprint arXiv:2210.02441* (2022).
[3] David Baidoo-Anu and Leticia Owusu Ansah. 2023. Education in the era of generative artificial intelligence (AI): Understanding the potential benefits of ChatGPT in promoting teaching and learning. *Journal of AI* 7, 1 (2023), 52–62.
[4] Som Biswas. 2023. Role of ChatGPT in Computer Programming. *Mesopotamian Journal of Computer Science* 2023 (2023), 9–15.
[5] Alasdair Blair and Samantha McGinty. 2010. It's good to talk? Developing feedback practice. *Gateway Papers* 1 (2010).
[6] Ali Borji. 2023. A categorical archive of chatgpt failures. *arXiv preprint arXiv:2302.03494* (2023).
[7] Julio C Caiza and Jose M Del Alamo. 2013. Programming assignments automatic grading: review of tools and implementations. *INTED2013 Proceedings* (2013), 5691–5700.
[8] Anita Carleton, Mark Klein, John Robert, Erin Harper, Robert Cunningham, Dionisio de Niz, John Foreman, John Goodenough, James Herbsleb, Ipek Ozkaya, Douglas Schmidt, and Forrest Shull. 2021. *Architecting the Future of Software Engineering: A National Agenda for Software Engineering Research & Development.* Accessed: 2023-Dec-7.
[9] Brenda Cheang, Andy Kurnia, Andrew Lim, and Wee-Chong Oon. 2003. On automated grading of programming assignments in an academic institution. *Computers & Education* 41, 2 (2003), 121–131.

[5] In practice, we found that students quickly identified any remaining false positives that slipped past GreAIter and the human graders since they were eager to improve their grades!

[10] Binglin Chen, Sushmita Azad, Rajarshi Haldar, Matthew West, and Craig Zilles. 2020. A validated scoring rubric for explain-in-plain-english questions. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*. 563–569.

[11] Eason Chen, Ray Huang, Han-Shin Chen, Yuen-Hsien Tseng, and Liang-Yi Li. 2023. GPTutor: a ChatGPT-powered programming tool for code explanation. In *International Conference on Artificial Intelligence in Education*. Springer, 321–327.

[12] Chase Geigle, ChengXiang Zhai, and Duncan C Ferguson. 2016. An exploration of automated grading of complex assignments. In *Proceedings of the Third (2016) ACM Conference on Learning@ Scale*. 351–360.

[13] Skyler Grandel, Douglas C Schmidt, and Kevin Leach. 2024. Applying Large Language Models to Enhance the Assessment of Parallel Functional Programming Assignments. In *Proceedings of the 1st International Workshop on Large Language Models for Code*. 102–110.

[14] Matthew Hertz. 2010. What do" CS1" and" CS2" mean? Investigating differences in the early courses. In *Proceedings of the 41st ACM technical symposium on Computer science education*. 199–203.

[15] Wayne Holmes, Kaska Porayska-Pomsta, Ken Holstein, Emma Sutherland, Toby Baker, Simon Buckingham Shum, Olga C Santos, Mercedes T Rodrigo, Mutlu Cukurova, Ig Ibert Bittencourt, et al. 2021. Ethics of AI in education: Towards a community-wide framework. *International Journal of Artificial Intelligence in Education* (2021), 1–23.

[16] Stephen C Johnson. 1977. *Lint, a C program checker*. Bell Telephone Laboratories Murray Hill.

[17] Anders Jonsson and Gunilla Svingby. 2007. The use of scoring rubrics: Reliability, validity and educational consequences. *Educational research review* 2, 2 (2007), 130–144.

[18] Anant Kharkar, Roshanak Zilouchian Moghaddam, Matthew Jin, Xiaoyu Liu, Xin Shi, Colin Clement, and Neel Sundaresan. 2022. Learning to reduce false positives in analytic bug detectors. In *Proceedings of the 44th International Conference on Software Engineering*. 1307–1316.

[19] Lucas Kohnke, Benjamin Luke Moorhouse, and Di Zou. 2023. ChatGPT for language teaching and learning. *Relc Journal* 54, 2 (2023), 537–550.

[20] Terry K Koo and Mae Y Li. 2016. A guideline of selecting and reporting intraclass correlation coefficients for reliability research. *Journal of chiropractic medicine* 15, 2 (2016), 155–163.

[21] Tiffany H Kung, Morgan Cheatham, Arielle Medenilla, Czarina Sillos, Lorie De Leon, Camille Elepaño, Maria Madriaga, Rimel Aggabao, Giezel Diaz-Candido, James Maningo, et al. 2023. Performance of ChatGPT on USMLE: Potential for AI-assisted medical education using large language models. *PLoS digital health* 2, 2 (2023), e0000198.

[22] Kenneth O McGraw and Seok P Wong. 1996. Forming inferences about some intraclass correlation coefficients. *Psychological methods* 1, 1 (1996), 30.

[23] Grégoire Mialon, Roberto Dessì, Maria Lomeli, Christoforos Nalmpantis, Ram Pasunuru, Roberta Raileanu, Baptiste Rozière, Timo Schick, Jane Dwivedi-Yu, Asli Celikyilmaz, et al. 2023. Augmented language models: a survey. *arXiv preprint arXiv:2302.07842* (2023).

[24] Mistral AI. 2024. *Mistral Large 2*. https://mistral.ai/

[25] Vu Nguyen, Sophia Deeds-Rubin, Thomas Tan, and Barry Boehm. 2007. A SLOC counting standard. In *Cocomo ii forum*, Vol. 2007. Citeseer, 1–16.

[26] OpenAI. 2022. *ChatGPT*. https://chat.openai.com/

[27] Michael Sheinman Orenstrakh, Oscar Karnalim, Carlos Anibal Suarez, and Michael Liut. 2023. Detecting llm-generated text in computing education: A comparative study for chatgpt cases. *arXiv preprint arXiv:2307.07411* (2023).

[28] Chris Parnin and Alessandro Orso. 2011. Are automated debugging techniques actually helping programmers?. In *Proceedings of the 2011 international symposium on software testing and analysis*. 199–209.

[29] James Perretta, Westley Weimer, and Andrew DeOrio. 2019. Human vs. automated coding style grading in computing education. In *2019 ASEE Annual Conference & Exposition*.

[30] Junaid Qadir. 2023. Engineering education in the era of ChatGPT: Promise and pitfalls of generative AI for education. In *2023 IEEE Global Engineering Education Conference (EDUCON)*. IEEE, 1–9.

[31] D Royce Sadler. 1989. Formative assessment and the design of instructional systems. *Instructional science* 18, 2 (1989), 119–144.

[32] Sami Sarsa, Paul Denny, Arto Hellas, and Juho Leinonen. 2022. Automatic generation of programming exercises and code explanations using large language models. In *Proceedings of the 2022 ACM Conference on International Computing Education Research-Volume 1*. 27–43.

[33] Johannes Schneider, Robin Richner, and Micha Riser. 2023. Towards trustworthy autograding of short, multi-lingual, multi-type answers. *International Journal of Artificial Intelligence in Education* 33, 1 (2023), 88–118.

[34] Johannes Schneider, Bernd Schenk, Christina Niklaus, and Michaelis Vlachos. 2023. Towards LLM-based Autograding for Short Textual Answers. *arXiv preprint arXiv:2309.11508* (2023).

[35] Nigar M Shafiq Surameery and Mohammed Y Shakor. 2023. Use chat gpt to solve programming bugs. *International Journal of Information Technology and Computer Engineering* 31 (2023), 17–22.

[36] Kehui Tan, Tianqi Pang, and Chenyou Fan. 2023. Towards Applying Powerful Large AI Models in Classroom Teaching: Opportunities, Challenges and Prospects. *arXiv preprint arXiv:2305.03433* (2023).

[37] Ahmed Tlili, Boulus Shehata, Michael Agyemang Adarkwah, Aras Bozkurt, Daniel T Hickey, Ronghuai Huang, and Brighter Agyemang. 2023. What if the devil is my guardian angel: ChatGPT as a case study of using chatbots in education. *Smart learning environments* 10, 1 (2023), 15.

[38] Eva AM Van Dis, Johan Bollen, Willem Zuidema, Robert van Rooij, and Claudi L Bockting. 2023. ChatGPT: five priorities for research. *Nature* 614, 7947 (2023), 224–226.

[39] Yaqing Wang, Quanming Yao, James T Kwok, and Lionel M Ni. 2020. Generalizing from a few examples: A survey on few-shot learning. *ACM computing surveys (csur)* 53, 3 (2020), 1–34.

[40] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. 2023. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. arXiv:2201.11903 [cs.CL]

[41] Jules White, Quchen Fu, Sam Hays, Michael Sandborn, Carlos Olea, Henry Gilbert, Ashraf Elnashar, Jesse Spencer-Smith, and Douglas C Schmidt. 2023. A prompt pattern catalog to enhance prompt engineering with chatgpt. *Proceedings of the 30th Conference on Pattern Languages of Programs* (2023).

[42] Jules White, Sam Hays, Quchen Fu, Jesse Spencer-Smith, and Douglas C Schmidt. 2024. Chatgpt prompt patterns for improving code quality, refactoring, requirements elicitation, and software design. In *Generative AI for Effective Software Development*. Springer, 71–108.

[43] Yuankai Xue, Hanlin Chen, Gina R Bai, Robert Tairas, and Yu Huang. 2024. Does ChatGPT Help With Introductory Programming? An Experiment of Students Using ChatGPT in CS1. In *Proceedings of the 46th International Conference on Software Engineering: Software Engineering Education and Training*. 331–341.

[44] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. ReAct: Synergizing Reasoning and Acting in Language Models. arXiv:2210.03629 [cs.CL]

[45] Yongchao Zhou, Andrei Ioan Muresanu, Ziwen Han, Keiran Paster, Silviu Pitis, Harris Chan, and Jimmy Ba. 2022. Large language models are human-level prompt engineers. *arXiv preprint arXiv:2211.01910* (2022).