WIP: How Effective Are LLM-Implemented Autograders for Programming Assignments Compared to Human Graders?

Kevin Lewis[*] and Hui Chen[†][*]
[*]*Department of Computer Science*, *CUNY Graduate Center*, 365 5th Ave., New York, NY 10016, USA
[†]*Department of Computer & Information Science*, *CUNY Brooklyn College*, 2900 Bedford Ave., Brooklyn, NY 11210
Email: klewis@gradcenter.cuny.edu, huichen@ieee.org

***Abstract*—This research-to-practice WIP paper describes the development and evaluation of a generative Large Language Model (gLLM)-based autograder for computer programming assignments. Manual grading is becoming increasingly unsustainable due to growing student enrollment and the demand for timely, high-quality feedback. To address these challenges, this study explores the use of automated grading tools to reduce instructors' workload and improve scalability. The proposed autograder takes a "reverse-engineering" approach, i.e., it converts student code into structured natural language summaries, which are then compared against predefined grading rubrics. An evaluation is performed using an external dataset (the Menagerie dataset), which contains real student submissions graded by four human graders. The objective is to assess the alignment between grades assigned by the autograder and those assigned by human graders. Findings indicate that the autograder closely matches human grading when letter grades are considered, though it performs less accurately with fine-grained numerical scores. While not yet a complete substitute for human assessment, the autograder shows strong potential as a scalable, efficient tool for supporting grading in programming education.**

***Index Terms*—Grades, Grading Systems, Automated Grading, Student Assessment, Computing Skills**

I. Introduction

Evaluating student-written code in programming courses traditionally requires significant human effort to ensure that assignments align with assignment specifications and requirements. Motivated by the need to give students timely feedback and to scale instruction for large classes, educators have long sought to automate this process through the development of autograding systems. These systems use a range of techniques, such as unit testing, static analysis, and machine learning [1]–[5]. Despite these advances, autograding systems often struggle to assess open-ended assignments, accommodate a great range of diversity in programming assignments, and evaluate the students' work beyond correctness or predefined patterns in the code [6], [7].

Generative Large Language Models (gLLMs) have become pivotal in harnessing artificial intelligence for a variety of purposes due to their demonstrated capability to "comprehend" natural language text and program code. These models are typically accessed via textual prompts that describe the desired task. A number of prompting strategies have been proposed, including few-shot prompting, chain-of-thought prompting, generated knowledge prompting, and graph-of-thought prompting [8]–[10]. These works suggest that gLLMs possess emerging reasoning capabilities [11], [12], raising hopes that they can be used to build systems that automate the evaluation of student-written code while addressing the challenges that traditional autograding systems face.

Despite this potential, several unanswered and challenging questions remain regarding how to effectively harness gLLMs for grading programming assignments. First, *what system design leveraging gLLMs is effective for generating grades for student-written programs*? Second, *to what extent can a gLLM-based autograder achieve the same level of accuracy as human graders*? These questions are particularly challenging because human graders often exhibit high variability in their grading; however, they are able to identify a wide range of issues in students' programs in addition to correctness [7], [13].

Thus, the focus of this work is to investigate the extent to which a gLLM-based grading system produces grades comparable to those of human graders. The design of the system and the study is guided by the Criterion-Referenced Assessment theory [14], [15]. The system is designed to evaluate students' work against a set standard derived from the assignment description rather than in comparison to each other. This work is also motivated by the theory of feedback, which emphasizes the importance of providing students with feedback that is not only accurate but also timely and relevant to their learning objectives [16]. Grades, in conjunction with descriptive comments, can serve as effective formative and summative evaluation for students' work [17]–[20].

To evaluate the autograder, we use a publically available dataset [13], which includes real student Java programming submissions from an introductory CS1 course and detailed grading feedback from four human assessors. The use of this dataset enables a robust comparison between the performance of the gLLM-based autograder and human graders.

The contribution of this work is twofold. First, leveraging gLLMs, we design a novel autograder via a "reverse engineering" approach, where the autograder first transforms the students' code into structured natural language summaries, representing a written specification of their programs, and then assesses it against a set of specific grading rubric criteria derived from the assignment. Second, we evaluate the performance of the gLLM-based grading system against a dataset of student submissions that have been graded by multiple human graders. The evaluation indicates that the gLLM-based grading system is comparable to human graders in terms of both accuracy and variability.

II. Related Work

There is a rich body of literature dedicated to the design and evaluation of automated grading systems in computing education [3], [7], [21].

A primary set of automated grading systems assesses the correctness of students' work using testing, such as unit testing, to provide grades and feedback [3], [7]. A particular challenge for instructors who adopt such systems is that they must provide a comprehensive suite of tests for each programming assignment, which requires significant effort and time. Students also face challenges with these systems, as they must ensure their submissions produce exact outputs for the test cases or guarantee that their implementation can compile and run with the test cases. A novel approach that lessens the burden on instructors for writing the test cases is Web-CAT [2], which requires all students to write and submit unit tests for their own code, and the system grades students based on the correctness of the tests written by their peers. Web-CAT has found success in large engineering programming courses that enroll a significant number of students; however, whether it can be effectively deployed in smaller classes remains an open question. More importantly, the approach suffers from weaknesses common to other testing-based grading systems, such as evaluating students' work solely on the correctness of program code, without sufficiently addressing other aspects of programming assignments, such as quality of design, style, and documentation.

Another approach to automated grading is to use static analysis tools to evaluate the quality of the code. These tools can check for common programming errors, adherence to coding standards, and other quality metrics. For grading students' work, these tools are used to assess the similarity of the students' submissions to the instructor's model solution [3], [7]. The similarity can be computed by examining numerous aspects, including but not limited to abstract syntax trees, data flow graphs, and control flow graphs [7]. A limitation of these tools is their inability to capture the dynamic behavior of programs, as testing does. In addition, their ability to check students' work is based on a predefined set of rules, which is not only often limited but can also produce incorrect warnings (i.e., false positives). Finally, similar to testing-based grading systems, building model solutions and writing new rules can be laborious.

In recent years, researchers have also explored the use of machine learning techniques to automate the grading process. These techniques can learn from historical data and identify patterns in student submissions, allowing for more accurate and personalized feedback. For example, some studies have used supervised learning algorithms to classify student submissions based on their quality and correctness [5]. However, these approaches often require large amounts of labeled student submissions and associated programming assignments, which can be difficult to obtain in practice. Moreover, the performance of these models can be sensitive to the choice of features and the quality of the training data [5].

The use of gLLMs in programming education has gained significant attention in recent years, particularly in the context of providing feedback to students [5], [22], [23], [23]–[34]. Our work explores the design of a gLLM-based autograder. gLLMs are typically trained on vast text corpora through unsupervised learning and can exhibit generative capabilities, allowing them to produce human-like text and understand complex programming concepts. A gLLM-based autograder has the potential to provide more nuanced and context-aware feedback compared to traditional systems. For instance, it can be extended to analyze the structure and semantics of code, identify potential bugs, and suggest improvements in design and style. As it matures, it could lead to a more comprehensive evaluation of students' work, addressing some limitations of existing grading approaches. In addition, unlike traditional machine learning-based systems, these models are already pretrained and typically used in a prompt-based manner, eliminating the need for extensive feature engineering and training on large labeled datasets. This makes them more accessible and easier to deploy in educational settings.

III. Generative LLM-based Autograder Design

We design an autograder system using gLLMs, denoted as $o = \mathcal{G}(a, r, s)$, where a is an assignment, r is the grading rubric for the assignment, s is a student submission for the assignment, and o is the system output – the grade of the student's submission. In this work, o is a numeric score on a scale of 0 to 100.

One advantage of our design is that it takes the grading rubric as input to the system. A grading rubric, in its basic form, is often presented as a matrix that provides levels of achievement for a set of criteria [35]. Despite its limitations [35], [36], rubric-driven assessment has two advantages for autograding. First, it breaks down evaluation criteria into specific, measurable components and ensures consistency and transparency [7], [24], [25], [37]. More importantly, since gLLMs heavily rely on prompt engineering, the grading rubric can be an indispensable source for formulating high-quality prompts to guide gLLMs.

Via an initial exploration, we settle on a system design that resembles a "reverse engineering" process. The system first generates a natural language summary of the program code submitted by the students, serving as a specification of the work, and then compares this summary against the grading rubric.

The process begins with the preprocessing of student submissions. First, the system checks whether the student's submission can compile and run. A common practice in programming education is to assign a failing grade to those submissions that fail to compile and run [13], [23]. Our system follows this practice and only proceeds with those submissions that can. Second, a student's submission will also be tagged and skipped for autograding if it is a verbatim copy of the starter code provided by the instructor.

The next step in our system is code summarization. To overcome the limited context window length of gLLMs [38], we break down the submitted code (s) into smaller chunks (c) according to the context window length of the gLLM used and the natural boundaries of the code segment, e.g., classes or methods, i.e., $s = c_1, c_2, \ldots, c_n$, where $|c_i| \leq L$, and L is the context window length of the gLLM used. Given a prompt p_i, the system generates a natural language summary (or digest, denoted as d_i) of the code chunk c_i as $d_i = \text{gLLM}(p_i, c_i)$.

Prompt p_i is designed to elicit a summary of the code chunk c_i that is relevant to the grading rubric and the assignment. It takes the form of: "`Given the {assignment description}, according to the {grading rubric}, summarize the chunk of the submission {code chunk}`", where "{x}" denotes a field that need to be filled in with actual value of "x". The summaries are then pooled together to form a complete summary of the student submission, i.e., $d = \{d_1, d_2, \ldots, d_n\}$.

Motivated by the generated knowledge prompting technique [39], in the same session with the gLLM, after generating the summary d, we also prompt the gLLM to generate a set of responses that can be used to evaluate the entire submission according to the grading rubric. This is done by asking the gLLM to provide a discrete `Yes/No` answer for each criterion in the grading rubric (denoted as $i_j \in 1, 0$ for criterion j).

For each criterion where the gLLM gives a `Yes` answer, the system further prompts the gLLM to provide two discrete scores. One score assesses the quality of the student's work relative to the grading rubric criterion and is in the range `Excellent/Good/Poor` ($l_j \in \{3, 2, 1\}$). The other score assesses the quantity of the student's work addressing the criterion and is in the range `All/Some/Little` ($t_j \in \{3, 2, 1\}$).

The system finally computes a score on the scale of 0 to 100 as follows: $o = 100 \sum_{j=1}^{N} (i_j l_j + i_j t_j)/(6N)$, where N is the number of rubric criteria.

IV. Evaluation and Results

A. Choice of gLLM

Our long-term goal is to develop an autograder that does not rely on a commercial gLLM and can be hosted by an educational institution to alleviate security and privacy concerns regarding student data. As a result, we elect to use LLaMA 3.1, specifically, the LLaMA 3.1 8B model, as the gLLM for experimenting with and evaluating our autograder.

B. Evaluation Dataset

The evaluation is based on a high-quality dataset collected and released by Messer et al. [13]. The dataset consists of a set of programming assignments from a programming course and 400 student submissions spanning multiple semesters. Each submission was graded independently by four human graders on a 100-point scale. As is often noted in the literature [24], [25], human graders exhibit variance in their grades. This dataset is no exception. As such, it is particularly well-suited for evaluating the autograder not only in terms of grading accuracy but also in terms of consistency when compared to multiple human graders.

C. System Design and Tuning

To arrive at suitable design parameters, such as prompt format and processing pipelines, we randomly partition the submissions into two sets: a validation partition and a test partition. The validation partition is used to tune prompt formats and other parameters, including the model's temperature and top-p sampling. Using the validation partition, we eventually settle on the design described in Section III.

D. Results

We use the autograder to grade the submissions in the test partition, which is unseen during the system design and tuning process.

To compare the grading accuracy of the autograder with that of human graders, we consider a score assigned by the autograder to be accurate when it falls within the range of scores assigned by the four human graders. By this definition, 70% of the submissions are accurately graded by the autograder.

To put this accuracy in context, we analyze the correlation between the grades assigned by the autograder and those given by human graders. It is well established that class rank is often more meaningful than absolute scores in educational settings [40]. Following this, we use a rank correlation method to evaluate grading accuracy. Specifically, we use Kendall's Tau, as it is generally considered more accurate, although computationally more expensive [41].

Figure 1 illustrates the rank correlation of submission scores among the autograder and four human graders using Kendall's Tau. The τ values between the scores given by the autograder and those given by the four human graders range from 0.15 to 0.26. Since the correlations among the four human graders range from 0.10 to 0.33, factoring in the p-values, we conclude that the correlation coefficients indicate a moderate level of agreement between the autograder and the human graders – the correlation is not as strong as the most consistent pair of human graders, but it is not worse than the pair of human graders with the greatest variance.

At higher education institutions in the U.S., letter grades are often used. As such, we convert the scores in a scale of 0 to 100 to a letter grades: A (90–100), B (80–89), C (70–79), D (60–69), and F (0–59) and then assign grade points to the submissions, where A=4, B=3, C=2, D=1, and F=0. Following this, we compare the machine grades with the 4 human graders' grades using Kendall's Tau rank correlation coefficients. The results are shown in Figure 2.

The observations are as follows. Human graders exhibit a high level of disagreement in their scores; however, the disagreement is significantly narrowed when we compare letter

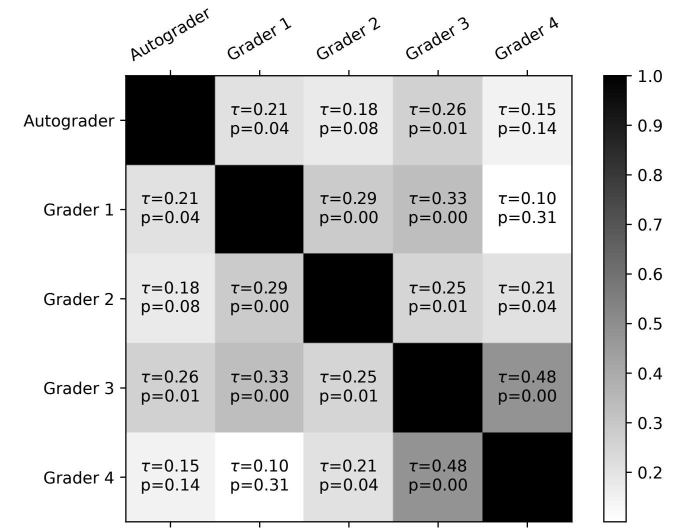

Fig. 1. Rank correlation of the grades (scores in the scale of 0 to 100) assigned by the autograder and by four human graders using Kendall's Tau.

grades. The τ values between the letter grades assigned by the four human graders range from 0.212 to 0.486 – much higher than the score-level correlations. Similarly, we observe that the autograder's letter grades are more consistent with those of the human graders than the raw scores. The τ values between the autograder and the human graders range from 0.384 to 0.494, suggesting a correlation strength comparable to that among human graders. These correlations also come with significantly improved p-values. This result suggests that the autograder achieves performance similar to that of human graders when used to assign letter grades of coarse granularity (i.e., A, B, C, D, and F), rather than fine-grained numerical scores (i.e., 0 – 100).

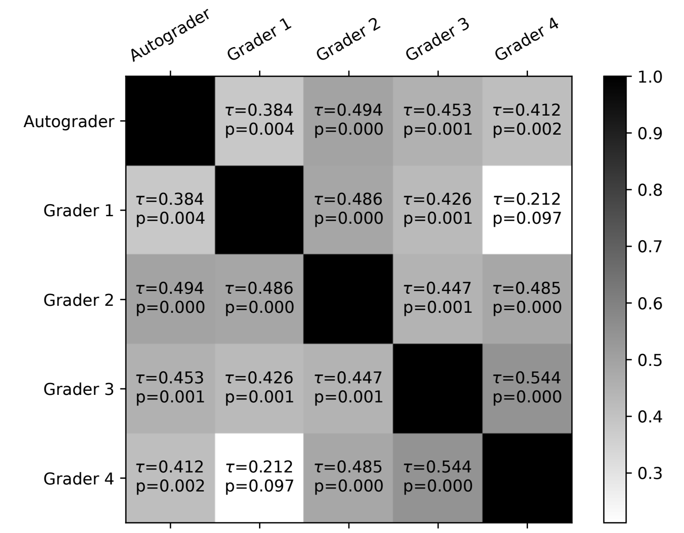

Fig. 2. Rank correlation of letter grades assigned by autograder and by four human graders using Kendall's Tau.

V. Conclusion and Future Work

In this Work-in-Progress paper, we explored the potential of using gLLMs to realize an automated grading system for programming assignments. Our exploration suggests an effective system design: an approach resembling "reverse engineering," where the system first generates a natural language summary of the program code—effectively a written specification—and then compares the summary against the criteria in the grading rubric. When fully realized, this system has the potential to overcome the limitations of traditional autograding methods, such as testing- and static analysis-based approaches, and can provide an assessment of student submissions that goes beyond code correctness and limited set of code patterns.

The evaluation using a high-quality dataset indicates that the gLLM-based autograder achieves grading accuracy comparable to human graders when comparing letter grades. The evaluation also reveals a limitation: it has yet to achieve the same level of consistency as human graders when comparing fine-grained numerical scores on a scale of 0–100.

To complete this work-in-progress study, future work includes the following directions:

1) In this paper, we used the open-source LLaMA 3.1 8B as the gLLM. While this model is powerful, it is not the best-performing gLLM currently available. To what extent could a stronger gLLM improve the grading accuracy of our system? To explore this question, we have begun experimenting with larger and newer versions of LLaMA. Future work also includes experimenting with models like DeepSeek and ChatGPT.
2) Prompt engineering methods significantly impact the reasoning performance of gLLMs. Can we enhance the autograder using advanced prompting techniques, such as Retrieval-Augmented Generation (RAG)?
3) We plan to enhance the autograder by providing both grades and descriptive feedback. This enhancement is particularly important. First, even human graders can exhibit significant variances in grading, as such there is an inherent limit we can improve autograder's grading accuracy and consistency. Second, research suggests that grades alone are not sufficient to motivate students to learn [20], [42]. With this enhancement, we aim to investigate whether an autograder that provides grades along with descriptive feedback can better support and motivate students toward self-regulated learning.

Acknowledgements

This material is based upon work supported by the National Science Foundation under Grant No. 2235976. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation. The authors are grateful to Lawrence Goetz of the Computer & Information Science Department at Brooklyn College for maintaining the departmental computational cluster that supported this work.

References

[1] R. Conejo, B. Barros, and M. F. Bertoa, "Automated assessment of complex programming tasks using SIETTE," *IEEE Transactions on Learning Technologies*, vol. 12, no. 4, pp. 470–484, 2018.

[2] S. H. Edwards, "Work-in-progress: Program grading and feedback generation with web-cat," in *Proceedings of the first ACM conference on Learning@ scale conference*, 2014, pp. 215–216.

[3] K. M. Ala-Mutka, "A survey of automated assessment approaches for programming assignments," *Computer science education*, vol. 15, no. 2, pp. 83–102, 2005.

[4] D. Liu and A. Petersen, "Static analyses in python programming courses," in *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, 2019, pp. 666–671.

[5] M. Messer, N. C. Brown, M. Kölling, and M. Shi, "Machine learning-based automated grading and feedback tools for programming: a meta-analysis," in *Proceedings of the 2023 Conference on Innovation and Technology in Computer Science Education V. 1*, 2023, pp. 491–497.

[6] B. Cheang, A. Kurnia, A. Lim, and W.-C. Oon, "On automated grading of programming assignments in an academic institution," *Computers & Education*, vol. 41, no. 2, pp. 121–131, 2003.

[7] M. Messer, N. C. Brown, M. Kölling, and M. Shi, "Automated grading and feedback tools for programming education: A systematic review," *ACM Transactions on Computing Education*, vol. 24, no. 1, pp. 1–43, 2024.

[8] S. Yao, D. Yu, J. Zhao, I. Shafran, T. Griffiths, Y. Cao, and K. Narasimhan, "Tree of thoughts: Deliberate problem solving with large language models," *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[9] M. Besta, N. Blach, A. Kubicek, R. Gerstenberger, M. Podstawski, L. Gianinazzi, J. Gajda, T. Lehmann, H. Niewiadomski, P. Nyczyk *et al.*, "Graph of thoughts: Solving elaborate problems with large language models," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 38, no. 16, 2024, pp. 17 682–17 690.

[10] L. LUO, Y.-F. Li, R. Haf, and S. Pan, "Reasoning on graphs: Faithful and interpretable large language model reasoning," in *The Twelfth International Conference on Learning Representations*, 2024.

[11] Y. Zhang, S. Mao, T. Ge, X. Wang, Y. Xia, W. Wu, T. Song, M. Lan, and F. Wei, "LLM as a mastermind: A survey of strategic reasoning with large language models," in *First Conference on Language Modeling*, 2024. [Online]. Available: https://openreview.net/forum?id=iMqJsQ4evS

[12] Z.-Z. Li, D. Zhang, M.-L. Zhang, J. Zhang, Z. Liu, Y. Yao, H. Xu, J. Zheng, P.-J. Wang, X. Chen, Y. Zhang, F. Yin, J. Dong, Z. Guo, L. Song, and C.-L. Liu, "From system 1 to system 2: A survey of reasoning large language models," 2025. [Online]. Available: https://arxiv.org/abs/2502.17419

[13] M. Messer, N. C. C. Brown, M. Kölling, and M. Shi, "How consistent are humans when grading programming assignments?" 2025. [Online]. Available: https://arxiv.org/abs/2409.12967

[14] J. M. Turnbull, "What is… normative versus criterion-referenced assessment," *Medical teacher*, vol. 11, no. 2, pp. 145–150, 1989.

[15] L. A. Bond, "Norm-and criterion-referenced testing," *Practical Assessment, Research, and Evaluation*, vol. 5, no. 1, 1996.

[16] A. Irons and S. Elkington, *Enhancing learning through formative assessment and feedback*. Routledge, 2021.

[17] V. J. Shute, "Focus on formative feedback," *Review of educational research*, vol. 78, no. 1, pp. 153–189, 2008.

[18] S. M. Brookhart, "Summative and formative feedback." in *The Cambridge handbook of instructional feedback*, A. A. Lipnevich and J. K. Smith, Eds. Cambridge University Press, 2018, pp. 52–78.

[19] T. R. Guskey, "Grades versus comments: Research on student feedback," *Phi Delta Kappan*, vol. 101, no. 3, pp. 42–47, 2019.

[20] K. Chamberlin, M. Yasué, and I.-C. A. Chiang, "The impact of grades on student motivation," *Active Learning in Higher Education*, vol. 24, no. 2, pp. 109–124, 2023.

[21] H. Keuning, J. Jeuring, and B. Heeren, "A systematic literature review of automated feedback generation for programming exercises," *ACM Transactions on Computing Education (TOCE)*, vol. 19, no. 1, pp. 1–43, 2018.

[22] T. W. Li, S. Hsu, M. Fowler, Z. Zhang, C. Zilles, and K. Karahalios, "Am i wrong, or is the autograder wrong? effects of ai grading mistakes on learning," in *Proceedings of the 2023 ACM Conference on International Computing Education Research-Volume 1*, 2023, pp. 159–176.

[23] F. Nilsson and J. Tuvstedt, "GPT-4 as an automatic grader: The accuracy of grades set by gpt-4 on introductory programming assignments," 2023.

[24] W. Yeadon, A. Peach, and C. P. Testrow, "A comparison of human, GPT-3.5, and GPT-4 performance in a university-level coding course," *arXiv preprint arXiv:2403.16977*, 2024.

[25] P. Oli, R. Banjade, J. Chapagain, and V. Rus, "Automated assessment of students' code comprehension using llms," *arXiv preprint arXiv:2401.05399*, 2023.

[26] T. Shin, Y. Razeghi, R. L. Logan IV, E. Wallace, and S. Singh, "Autoprompt: Eliciting knowledge from language models with automatically generated prompts," *arXiv preprint arXiv:2010.15980*, 2020.

[27] O. Fagbohun, N. Iduwe, M. Abdullahi, A. Ifaturoti, and O. Nwanna, "Beyond traditional assessment: Exploring the impact of large language models on grading practices," *Journal of Artificial Intelligence and Machine Learning & Data Science*, vol. 2, no. 1, pp. 1–8, 2024.

[28] C. Geng, Z. Yihan, B. Pientka, and X. Si, "Can ChatGPT pass an introductory level functional language programming course?" *arXiv preprint arXiv:2305.02230*, 2023.

[29] M.-D. Popovici, "ChatGPT in the classroom. exploring its potential and limitations in a functional programming course," *International Journal of Human–Computer Interaction*, pp. 1–12, 2023.

[30] R. Balse, V. Kumar, P. Prasad, and J. M. Warriem, "Evaluating the quality of LLM-generated explanations for logical errors in cs1 student programs," in *Proceedings of the 16th Annual ACM India Compute Conference*, 2023, pp. 49–54.

[31] C. Singh, J. X. Morris, J. Aneja, A. M. Rush, and J. Gao, "Explaining patterns in data with language models via interpretable autoprompting," *arXiv preprint arXiv:2210.01848*, 2022.

[32] S. Wang, T. Xu, H. Li, C. Zhang, J. Liang, J. Tang, P. S. Yu, and Q. Wen, "Large language models for education: A survey and outlook," *arXiv preprint arXiv:2403.18105*, 2024.

[33] R. Tufano, L. Pascarella, M. Tufano, D. Poshyvanyk, and G. Bavota, "Towards automating code review activities," in *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 2021, pp. 163–174.

[34] D. Cambaz and X. Zhang, "Use of AI-driven code generation models in teaching and learning programming: a systematic literature review," in *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1*, 2024, pp. 172–178.

[35] R. J. Howell, "Grading rubrics: Hoopla or help?" *Innovations in education and teaching international*, vol. 51, no. 4, pp. 400–410, 2014.

[36] E. Panadero and A. Jonsson, "A critical review of the arguments against the use of rubrics," *Educational Research Review*, vol. 30, p. 100329, 2020.

[37] S. Parihar, Z. Dadachanji, P. K. Singh, R. Das, A. Karkare, and A. Bhattacharya, "Automatic grading and feedback using program repair for introductory programming courses," in *Proceedings of the 2017 ACM conference on innovation and technology in computer science education*, 2017, pp. 92–97.

[38] X. Wang, M. Salmani, P. Omidi, X. Ren, M. Rezagholizadeh, and A. Eshaghi, "Beyond the limits: A survey of techniques to extend the context length in large language models," in *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence (IJCAI-24) – Survey Track*, 2024, pp. 1–9.

[39] J. Liu, A. Liu, X. Lu, S. Welleck, P. West, R. Le Bras, Y. Choi, and H. Hajishirzi, "Generated knowledge prompting for commonsense reasoning," in *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2022, pp. 3154–3169.

[40] P. T. Wangerin, "Calculating rank-in-class numbers: The impact of grading differences among law school teachers," *J. Legal Educ.*, vol. 51, p. 98, 2001.

[41] A. R. Gilpin, "Table for conversion of kendall's tau to spearman's rho within the context of measures of magnitude of effect for meta-analysis," *Educational and psychological measurement*, vol. 53, no. 1, pp. 87–92, 1993.

[42] A. A. Lipnevich, T. R. Guskey, D. M. Murano, and J. K. Smith, "What do grades mean? variation in grading criteria in american college and university courses," *Assessment in education: Principles, policy & practice*, vol. 27, no. 5, pp. 480–500, 2020.