Large Language Model Performance in Automatic Assessment on an Introductory Programming Course

Juuso Rytilahti[1], Erkki Kaila[1] and Valtteri Ingman[1]

[1]*University of Turku, Turku, Finland*

Abstract

Large language models (LLMs) are a potential solution for solving the significant assessment load in big courses with numerous assignments. However, the quality of the automated assessment may not match the evaluation by teachers or other experts. In this paper, we examine the automated assessment of programming-related tasks in a large-scale introductory programming course. The study is structured into two parts: first, we examine how reliably LLMs can assess the tasks compared to course teachers when provided the same rubric. Second, we try to find out if a simple autonomous agent pipeline, mimicking a review board, can improve the assessment outcome. The study was conducted on a university-level introductory Python course with more than 500 students. We chose a total of four programming-related assignments from the four final weeks of the course. First, we provided the selected LLM models with the student answers accompanied by an evaluation rubric and a simple prompt and recorded the resulting scores and feedback comments. Second, we built a pipeline of autonomous agents with different roles of a review board and used the student submissions as input for the pipeline, again recording the scores and comments. In the article, we discuss the feasibility and the performance of the given approaches. We also provide a detailed analysis of the comparison between the results of the two approaches and the teacher-assessed results, and discuss the differences in the results and the likely reasons for them. Finally, we outline the potential for future work.

Keywords

introductory programming, automatic assessment, artificial intelligence, large language models, autonomous agents

1. Introduction

The performance of large language models (LLMs) has increased significantly in a short time. In addition to this, the use of autonomous agents [1] as a way to increase the LLM's performance and abilities across various tasks has become more popular. This performant technology offers possibilities of easing teachers' workload on courses with a massive amount of participants, hopefully allowing teachers to allocate their time more efficiently to other important pedagogical tasks.

The need for automating assessment [2] to reduce teacher's workload on courses consisting of hundreds, or even thousands of students is high. Of course, fully automated assessment of any tasks is a challenge, and improbable to be achieved with the current tools soon. Still, automating even some parts of the assessment process would significantly reduce teachers' workload and free their time for more constructive teaching tasks.

This paper reports a research setup where LLM in different settings was used for automatically assessing introductory programming course assignment submissions. The purpose is to find out what kind of differences there are in the review results of different LLM reviewers and human reviewers. As a bigger goal, we are trying to find out whether it is possible to automate the review process, and if not, what are the biggest obstacles preventing us from doing so.

This paper is structured as follows: First, we present the background and some related work about using AI in evaluating student work. In the third section, we present the context, including the course and the assignments evaluated. in the fourth section, the research setup is presented, including the cost analysis of utilizing LLM tools and the preparation of data for analysis. The results are presented in

TKTP 2025: Annual Doctoral Symposium of Computer Science, 2.—3.6.2025 Helsinki, Finland

jubery@utu.fi (J. Rytilahti); ertaka@utu.fi (E. Kaila); valtteri.b.ingman@utu.fi (V. Ingman)

https://rytilahti-juuso.github.io/meet-the-researcher/ (J. Rytilahti)

0009-0002-8269-5645 (J. Rytilahti); 0000-0002-2407-9492 (E. Kaila)

© 2025 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

CEUR-WS.org/Vol-4181/paper02.pdf

CEUR Workshop Proceedings ceur-ws.org ISSN 1613-0073

published 2026-03-12

Section 5 and discussed in the sixth Section, where we also present the limitations of the current work. Finally, a conclusion and ideas for future work are presented.

2. Background and Related Work

Automatic assessment in programming education has great potential benefits. As Paiva et al. [2] states, automatically assessing the tasks enables students to receive immediate feedback from all their submissions and hence improve the learning outcome by trial-and-error approach. While there are successful examples of using large language models in programming education, for example in exercise generation [3], code generation [4], or as a programming assistant [5], the full potential of models in evaluating programming assignments is not yet fulfilled.

Estévez-Ayrez et al. [6] studied the usage of LLM tools in generating automatic feedback on students' programming submissions, but concluded that at this point, the quality is not good enough. Bengtson et al. [7] tested how reliably LLM can detect injected errors in submissions and concluded that the generated feedback seems consistent but LLM is not reliable in recognizing errors. Unit testing and automated testing frameworks are a more traditional method of automatically assessing student answers [8]. Gabbay et al. [9] found out, that while LLM is not yet capable of replacing such frameworks in courses, it can complement the functionality.

One of the biggest limitations of using LLMs is a phenomenon called "hallucination. Xu et al. [10] defined hallucination as a situation "*where the models generate plausible but factually incorrect or nonsensical information*". Additionally, they pointed out that hallucination is an innate limitation of the LLMs. While hallucinations are typically addressed to LLMs' balance between creativity and factuality, and as such, could be fixed with access to external knowledge, this is not true in all cases [11]. Hence, it seems that hallucination remains an unsolved problem as of now.

Jiang et al. [12] noticed that irrelevant context can significantly hinder even state-of-the-art LLM performance (e.g. o1-preview, GPT-4o). They also tested carefully problems requiring more logical thinking and noted that LLM logic capabilities seem superficial, often affected by e.g. changing premises or entities present in classical problems. Furthermore, minor alteration of choices, for example, changing their order or the way they are written affects how well the different LLMs do in the benchmarks, which are often cited as the best way for guiding the model selection [13].

There are some general ways of trying to improve LLM performance [14]. An LLM can either be fine-tuned with a small data set, or the input can be in a zero-shot or a few-shot setting. In fine-tuning, a model is trained further with a more specific data set related to the given problem. In a zero-shot setting, an LLM is given an input without any examples. In a few-shot setting, the input includes a few examples of the task the model is asked to perform. Lee et al. [15] underline the importance of elaborate prompting and "*advanced prompt engineering techniques*", but warn that using advanced prompting directly may result to suboptimal results.

It should be noted that the temperature setting of the LLM has a great effect on the produced output, as it changes the probability distribution (randomness) of the next token. Quite often it is argued that the lowest temperature is the best, as it gives the most consistent result. However, one could argue that this might not always lead to the best possible results. For example, in coding generation utilizing adaptive temperature seems to improve results when high temperature is used for challenging tokens and low temperature for confident tokens [16]. On the other hand, some studies indicate that temperature has little or no effect on the problem-solving results, see for example [17].

The usage of autonomous agents as part of the review process or automatic assessment has been explored in different settings. [18] explored the usage of autonomous agents as a way to provide students with interactive feedback on argumentative essays. They used the temperature setting of 0.2 and aligned two used personas so that one was inclined to more positive feedback, whereas the other was aligned to produce more negative feedback. Chan et al. [19] created *ChatEval*, a tool where LLM-based agents debate on the given question, and found that the multi-agent approach performed better than single prompt approaches. They propose three different abstract approaches, "One-By-One", where the agents are in a direct pipeline with full previous chat history given as context, "Simultaneous-Talk", where agents talk are addressed asynchronously, and "Simultaneous-Talk-with-Summarizer" where at the end of each iteration the debater agents receive a summary of the previous iteration. Our approach in this paper is similar to One-By-One.

3. Course Overview

The course under review is called Fundamentals of Programming and is quite a typical 7-week introductory programming course (often referred to as CS1 course). The programming language used is Python, and the course contains the fundamental concepts of the imperative programming paradigm, such as variables and expressions, conditional and repetitive statements, functions, I/O, and data structures. Some additional Python features, such as the use of the Pygame [20] library are included in the final weeks to keep the course motivating.

The course consists of lectures, tutorial exercises, demonstration assignments, and a final exam. The tutorial exercises are fully automatically assessed by an education tool called ViLLE [21]. The demonstration assignments, however, are assessed by course instructors. As so, these are the assignments discussed in this paper. There were a total of eight demonstration assignments, delivered in four final weeks of the course (two assignments per each week). The topics of the assignments were:

- `Week 1`: In the first week, the tasks were quite straightforward, as the students were also subject to learn the use of the GIT version control system and the VSCode editor [22]. In the first task, the students needed to complete a simple calculator program. In the second task, they were given a program consisting of several string manipulation functions with missing implementation and were asked to complete the program.
- `Week 2`: In the second week, the tasks were about finding and correcting errors in the program code. The first program was a number-guessing game and the second one a library application. Both programs contained syntax errors as well as several logical errors.
- `Week 3`: The third week was about planning and commenting programs. In the first task, the students were required to plan a program for keeping track of expenses. The plan should include function signatures and docstrings [23] that comment on their functionality. In the second task, the students were asked to familiarize themselves with the existing program modeling a warehouse and extend the program with a new feature while following the coding conventions.
- `Week 4`: The fourth week introduced an external library called Pygame [20] which is meant for creating multimedia applications and 2d games. In the first task, the students were asked to write a program that displays a snowman that can be controlled via keyboard. In the second one, there were images of three different-sized snowflakes given and the task was to write a program that creates a randomized snowfall on screen.

The first two weeks of assignments were presumably easier to assess as the answers could be mostly verified programmatically. The tasks in the third week were more open-ended with several acceptable solutions. The fourth week was also assumed to be more difficult to assess, as the solutions were at least partly visually verified.

4. Research Setup

The pipeline for research was constructed with Python programming language, utilizing LangChain [24]. The rubrics, exercise description, and student submission were provided in Finnish. However, we decided to provide the system prompts in English instead of Finnish to reduce the token costs. The three distinct roles assigned to the LLM to assess the exercises were the following:

1. `Reviewer`: The model was instructed as being a teacher and was asked to provide the score and detailed feedback on the submission. The model was then provided with the exercise description,

Table 1
The detailed breakdown of the token amounts of the different phases. Total amount of assessed assignments for GPT-4o was 3076 and o1-preview 311. The assessed assignment counts (3076 and 311), and their respective token counts also include submissions redacted from the analysis.

Step	Input	Output	Total
Reviewer	5,734,595	1,541,406	7,276,001
Commenter	7,761,163	1,131,694	8,892,857
Chair	9,329,648	421,461	9,751,109
Reviewer o1	307,550	999,417	1,306,967

rubric, and student submission in a separate message. If necessary, additional material was included (such as the example program to be fixed in some tasks).

2. `Commenter`: The model was instructed as being part of a reviewer board. The model was provided the chat history from the Reviewer role. The model was then asked to provide commentary and update the score if needed.
3. `Chairperson`: The model was instructed as being a chairperson of a reviewer board. Earlier review and comments made by the model were provided and the model was asked to provide the final score and comments.

For the Commenter and Chairperson roles, a full chat history of the previous steps was provided as initial input. The prompts are displayed in the Appendix A. All submissions were submitted through all three different reviewer roles. The temperature of all of the autonomous agents was 0.2, and the used model was GPT-4o (gpt-4o-2024-08-06). In addition, we also submitted a small subset (N=100) of the first-week exercises to the GPT o1-preview model (o1-preview-2024-09-12) to gain a reference and see the performance difference in state-of-the-art reasoning and a more traditional, foundational LLM that is generally used. With o1, only the Reviewer role was used.

The rubric used for each week was the same as the rubric used to instruct the human reviewers. This led to some unforeseen difficulties: for example, in weeks 1, 2, and 3, an abbreviation was used to denote the maximum score. In some cases, the model got the score scaling wrong. Hence, we needed to exclude some of the reviews from the final analysis due to incomparable grading.

4.1. Cost-Analysis

With the main analysis, a total of 22,8 million input tokens and 3,5 million output tokens were used with GPT-4o. At the time of the analysis, this led to a combined cost of $\approx$ 60€ (or \$61.50). The subset of the first-week tasks submitted to the o1 model cost 0.3 million input tokens and 1 million output tokens, which led to a total cost of $\approx$ 126.5€ (\$130). A more detailed breakdown of the token costs can be seen in Table 1. The costs for pricing were calculated utilizing OpenAI's pricing page [25].

It should be noted that the cost can greatly vary depending on the selected model. Additionally, the tokenizers used can and will greatly affect the cost-effectiveness as well. Furthermore, although submissions were code files, most of them, including rubrics and exercise tokens, were written in Finnish. This increases the counted token amount significantly as the tokenizer OpenAI uses is optimized mainly for English instead of Finnish.

4.2. Data Preparation

Before the analysis started, we removed all submissions from students who had not explicitly given a research permit. After this, the remaining data was pseudonymized by replacing the usernames with randomly generated numbers throughout the data. These code numbers allowed us to connect the reviews from different reviewer roles without dealing with personal information.

Before the analysis, we also needed to remove some inconsistent data. As a general rule, for each submission, this meant the data rows where the analysis was incomplete were erased from the data.

Table 2
The number of submissions reviewed for each week.

Week	Manual and all three LLM roles	Reference (o1)
Week 1	312	100
Week 2	340	-
Week 3	224	-
Week 4	173	-

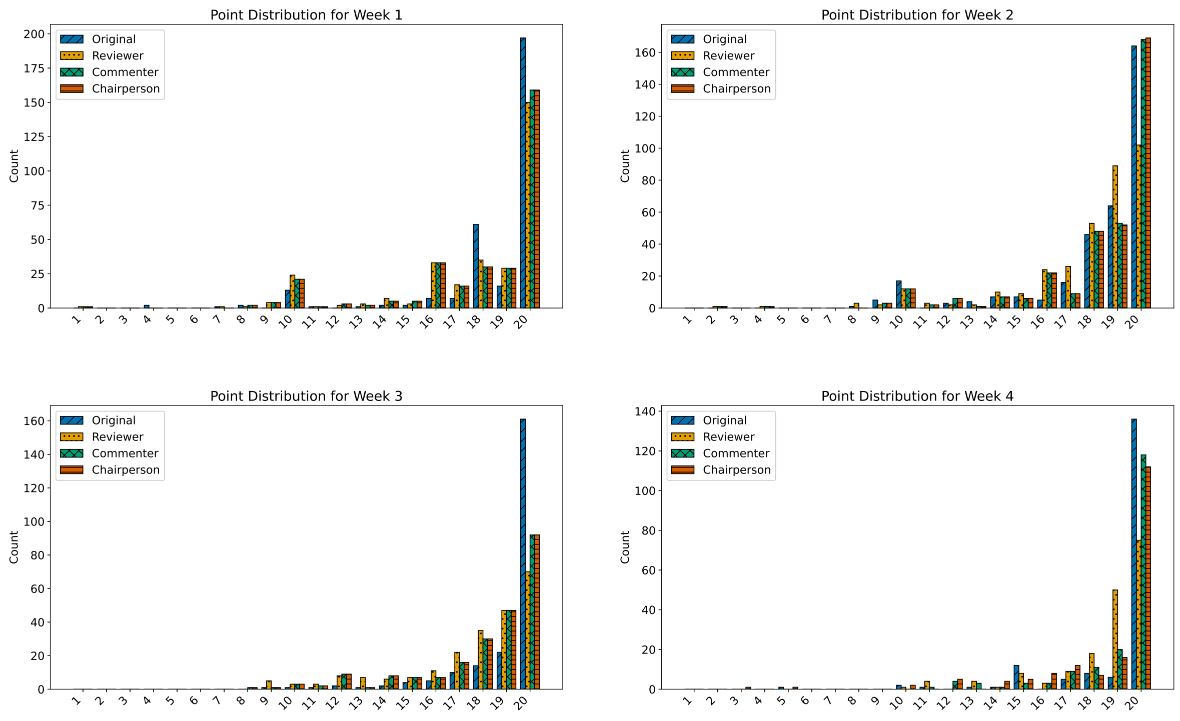

Figure 1: The point distribution for each week after data cleaning described in 4.2. The automatically assessed reviews are done by GPT-4o.

Typically, these cases arose because the submission had not been reviewed by a human, either due to late submission or suspicions of academic dishonesty. Some submissions were not reviewed by any of the models or the results were in unusable form (see Section 4). The N between different tasks (assignments) variates also because the students did not complete as many tasks at the end of the course as they did in the beginning.

The total number of submissions reviewed for each week is displayed in Table 2. In addition to running all submissions through GPT-4o via different roles in the pipeline, we picked a hundred first submissions from week 1 and ran them through the newest (at the time of writing) Chat GTP model, o1. This was done to see if there are noticeable differences between the models, providing guidelines for the future. Due to drastically higher costs, using the newest model for all submissions was not seen as a generally usable option at the time of writing the article.

Since the human instructors provided a combined score for each week (a single integer between 0 and 20 for both tasks), the comparison must be done week-by-week instead of comparing individual tasks. Still, the two tasks each week were quite similar which means that the comparison can still provide valid results.

Table 3
Average score obtained for each week by human reviewer and all LLM evaluation roles. The maximum score available was 20.

Week	Manual	Reviewer	Commenter	Chair
1	18.6	17.7	17.8	17.8
2	18.2	17.8	18.2	18.2
3	19.2	17.7	18.1	18.1
4	19.1	18.6	19.1	18.6

Table 4
Amount of scores that were smaller than, equal to, or larger than human-graded scores for each role each week.

Week	Reviewer			Commenter			Chair		
	Lower	Equal	Higher	Lower	Equal	Higher	Lower	Equal	Higher
1	37.0 %	51.8 %	11.2 %	35.4 %	51.8 %	12.8 %	35.4 %	51.8 %	12.8 %
2	43.1 %	39.6 %	17.3 %	18.5 %	59.5 %	22.0 %	18.2 %	59.8 %	22.0 %
3	58.5 %	35.2 %	6.3 %	45.1 %	47.3 %	7.6 %	45.1 %	47.3 %	7.6 %
4	42.8 %	44.5 %	12.7 %	22.0 %	62.4 %	15.6 %	27.7 %	62.5 %	9.8 %

5. Results

Table 3 displays the average score for each evaluation type, including the human instructors and all three roles of LLM evaluation for each week. The Figure 1 displays the point distribution for each week.

As seen in the table, the weekly average scores are quite close to each other. The commenter role seems to be closest to the human evaluators each week, but the differences are small. The Commenter and Chairperson points differ only in week 4, at other weeks they are equal. To further observe the differences, we performed a Mann-Whitney U-test between the human graded points and all LLM evaluation roles. Based on the tests, in almost all weeks all LLM role scores are statistically significantly different from human reviewer scores ($p < 0.05$). The only exception is the 2nd week, where Commenter and Chairperson roles were not significantly different from manually graded scores.

Next, we can observe the differences by calculating the number of scores that were smaller than, equal to, or larger than the human-graded scores for each role each week. This data is displayed in Table 4.

As seen in the table, the similarity percent varied between different weeks and roles. The Commenter and Chairperson role seemed to get better percent in almost all cases, the difference was greatest in weeks two and four. In the first week, the role did not seem to have any effect on the number of exactly correctly scored answers, while there still was a small difference in average scores Interestingly, in almost all cases, if the scores did not match, the LLM mostly awarded less points than the humar reviewer. The only exception is week 2, where Commenter and Chairperson roles provide higher points more frequently.

To observe the distance to human-graded scores, we also calculated the similarity percentages with two thresholds: within 1 points (meaning that answer was considered similar if the absolute difference between the scores was no more than 1 point) and within 4 points. The results are displayed in Table 5.

As seen in the table, the threshold values increase the percentages, as expected. In the second and the fourth week the threshold of 4 points provides quite high values.

Next, we calculated the amount of full-point (20 of 20) answers matching the human evaluation scores for each role. This was done to find out how often the different roles agree with the human reviewer that there is nothing wrong with the submission. The results are displayed in Table 6.

Based on these observations, it seems that in many cases where the human reviewer has provided the submission full points, the LLM reviewer has found something wrong with it. The Commenter and Chairperson roles performed closer to human reviewer in all cases, but still there is a considerable

Table 5
Amount of scores that were equal to human-graded scores for each role, and the ones that were equal in threshold of 1 or 4 points, respectively.

Week	Reviewer			Commenter			Chair		
Threshold	Equal	± 1	± 4	Equal	± 1	± 4	Equal	± 1	± 4
1	51.8 %	64.1 %	91.0 %	51.8 %	64.1 %	92.6 %	51.8 %	64.1 %	92.6 %
2	39.6 %	75.6 %	97.1 %	59.5 %	78.2 %	97.4 %	59.8 %	78.2 %	97.4 %
3	35.2 %	58.9 %	89.3 %	47.3 %	67.4 %	92.0 %	47.3 %	67.4 %	92.0 %
4	44.5 %	74.6 %	94.2 %	62.4 %	77.5 %	93.1 %	62.5 %	76.3 %	93. 1%

Table 6
Amount of times the roles matched the full point review (20 out of 20) by a human reviewer.

Week	20/20 manual (N)	Reviewer	Commenter	Chair
1	197	65.5 %	68.0 %	68.0 %
2	164	43.3 %	74.4 %	75.0 %
3	161	41.0 %	53.4 %	53.4 %
4	136	50.0 %	75.7 %	75.7 %

Table 7
Reference values obtained by running 100 week 1 submissions through GPT o1. The rows in order are: Average score of all 100 submissions, Number of full points matching human evaluators' full points, Amount of all scores matching human evaluator, and Amount of all scores matching human evaluators in threshold of <2 and <5 points, respectively.

Statistic	Manual	Reviewer	Commentor	Chair	Reference(o1)
Average score	18.7	18.0	18.2	18.2	17.9
Full points matched	64	40	44	44	49
Full points matched %	100 %	62.5 %	68.8 %	68.8 %	76.6 %
Matching scores	100 %	51 %	53 %	53 %	67 %
Matching, threshold <2	100 %	61 %	62 %	62 %	76 %
Matching, threshold <5	100 %	93 %	96 %	96 %	93 %

gap in amounts. Partly the point difference between roles Reviewer and Commenter can be explained by the pre-processing of the answers, because of a need to add some additional curly brackets in the submissions before review (see the Appendix A for reference), which sometimes lead the model to reduce points even when ignoring this edge case was included in the prompt of the model. However, this only explains a small amount of the detected deviation in the point distribution.

Finally, one hundred submissions to week 1 exercises were ran through the newest Chat GPT model, o1. The comparison to human reviewer and LLM roles using the older model, GPT 4o, are displayed in Table 7.

Based on the data, it seems that the newest model outperforms the old model in similarity to human reviewers in most categories. However, it should be noted that only 100 submissions from one week were used for comparison.

6. Discussion

The results show, that while in some conditions the LLM reviewers can get close to human reviewers, there is still a considerable gap between them. The average scores were quite close, but the U-test revealed statistically significant differences between the human reviewer scores and the LLM scores. Within a one-point threshold, the best-performing LLM roles could get quite close to human reviewers, and with a four-point threshold even closer. Still, in some tasks, roughly ten percent of tasks were graded differently even when considering the 4 point threshold. This indicates, that replacing human reviewers with LLM counterparts requires more work.

Even though the rubrics for the assignment were quite detailed, there is always a human factor involved in the review process. It should be noted that part of the difference in LLM vs. human review may be caused by the fact that, in the vast majority of cases, each submission was assessd by only one teacher. However, this effect may be in this particular case a little diluted, since there were a total of 15 reviewers. Still, we can say that this likely leads to a more spread in the grades given by a human reviewer.

Different methods to improve the automatic assessment utilizing LLMs should be considered as well. For example, instead of asking LLM to give a total score by decreasing points from the full score based on errors, we could try asking LLM to provide a checklist of errors. This could improve the performance of the LLM. It is also noteworthy that our experiment was conducted without providing the model with any examples of already graded exercises (zero-shot). As mentioned in Section 4, the model was given the exercise description, rubric and the submission to be evaluated. Providing previously-graded submissions and/or reference answers could potentially increase the quality of the grading.

One could argue that flexibility is the greatest asset that LLMs can offer in automated assessment in introductory programming education. Unit tests provide a way to measure the student's progress, but they also limit the possible exercise types and scope. If an LLM can reliably assess the work of a student, it could broaden the variety of possible exercise types even on massive online courses. This would also enable scoring of partially completed answers that may not even compile — traditional unit testing is typically unable to score such cases. Moreover, concepts such as program design, code quality, and commenting are usually difficult or impossible to assess with unit testing frameworks. Novice students often struggle with programming language syntax [26], which also means that providing them feedback automatically can be difficult with traditional methods.

The usage of the Open AI's newest model, o1, seemed promising. With the subset of the first-week task, the model seemed to achieve results that were closer to human reviewers than the older, generally used model. However, there is a big difference in cost-effectiveness as well, and as such, using the newest model would probably not be possible in most of the courses at the moment due to higher costs. Still, this seems to indicate that while the models get better, it should be possible to assign more and more review duties to them.

When the individual submission reviews were inspected, we found that in some cases LLM noticed an error in the submission that a human reviewer had missed. On the other hand, in some cases, LLM reviewers assessed the exercises wrong.

When delving deeper into the exercises for a more detailed analysis, we noticed that a common example of LLM falsely reducting points was when the students utilized Python's flexible nature. For example, LLM claimed that a string input of "+2" cast into an integer would cause an error, and the correct way to do it would be to parse the number without the plus sign. In reality, the parsing works without an issue. This kind of problem could probably be solved by giving the LLM access to a code compiler. In the string manipulation functions, LLM noticed that some of the exercises did not refer to a correct parameter variable given to a function, but instead directly modified the original global variable. In some cases, this was missed by the course personnel evaluating the submission.

In the third week, there are cases where the human reviewer has given full marks even though the specifications on the rubric are not fully fulfilled: for example, in the task requiring function signature only, some students had started implementing a given program as well. If partial implementation was done, human reviewers did overlook some missing specification present in the provided rubric, but the LLM reviewers considered it a mistake. Other challenges on the third-week assignments were the problems in defining the return types - for example, LLM reviewers considered defining None as a return type an error, as rubric mentioned defining None as a return type was not necessary to gain full marks, while human reviewers followed the rubric.

In the Pygame exercises, the visual elements provided some problems. For example, the appearance of the snowman was not explicitly stated in the rubric, and the LLM deducted points for example if the snowman was missing eyes (which were not explicitly stated to be included in the instructions).

Furthermore, in the snowflake assignment, LLM made some incorrect point reductions due to a misunderstanding that the requirement of differing size images would mean scaling them programmatically, while instead, the goal was to utilize the images of different size snowflakes provided with assignment.

Many of the problems listed above could have been avoided by providing more detailed descriptions in the rubric. It should also be noted that there were cases where the review provided by LLM was more accurate than the human counterpart. As seen in the results, LLM can achieve quite high performance, with some common mistakes made by LLM only revealed after inspecting many different submission reviews. For this reason for any future endeavors, we recommend first running a large subset of the exercises through the pipeline, then inspecting more closely the reviews where the human review differs, paying close attention to the rubric, and trying to ensure the consistent quality of human reviews.

6.1. Limitations

Due to reviewers giving a combined score for each week instead of scoring the individual exercises, we were not able to compare the scores exercise by exercise. It is likely, that if inspected separately, the difference in the marks given by LLM roles and the human reviewers would probably appear smaller. Still, even more rigorous testing is needed, as the achieved performance is quite reliant on the used model. Additionally, as the used language was Finnish, this will likely have had a negative impact on the performance, as the vast majority of the training data of the used model is written in English instead of Finnish.

Each submission was run through the evaluation pipeline only once using a low temperature (0.2) to provide more consistent results. Utilizing different settings could and likely would affect the performance of the used models. Furthermore, it should be noted that the submissions were given into the pipeline in a zero-shot setting. In many different tasks giving the model examples, as well as reference answers would likely improve the performance of the method. Giving examples in input often improves the performance of the models across various tasks [14] [27].

7. Conclusion and Future Work

Even though the model did not have access to code execution or visual output of programs, the LLM did quite well in the automatic assessment. The results and faced challenges highlight the need for rigorous testing before a larger data set is run through a pipeline that utilizes LLMs. Still, there are potential benefits in automating the assessment, for example, the possibility to offer students immediate feedback on their submission and the possibility to fix the mistakes before moving on to the next tasks. The largest challenges in giving students direct access to a similar system are the related costs, as well as the LLM's inability to reliably identify a full mark answer.

Future work should include approaches to minimizing costs. In the case of code submissions, it could be beneficial to first translate the submissions and all related material into English. Naturally, the performance in such cases would then be also dependent on the translation process. Another option would be to train and/or use models that are already optimized for the given language. The better results of the limited testing of the newest Open AI model also seem promising, but the larger-scale testing and general usage would require lower costs.

Exploring more diverse strategies in how agents communicate, as well as experimenting with various temperature settings should be examined in future work. Additionally, expanding the roles of the agents to incorporate other roles could be beneficial. Similarly, producing justifications using higher temperature settings and then the final score using lower temperatures could work as well. Finally, expanding the current work by providing the LLM access to code execution, as well as visual outputs of the executed programs could be an interesting approach.

Acknowledgments

This work has been supported by FAST, the Finnish Software Engineering Doctoral Research Network, funded by the Ministry of Education and Culture, Finland.

Declaration on Generative AI

During the preparation of this work, the author(s) used OpenAI's different models, e.g. GPT-o1 as a helpful tool for creating the pipeline. After using these tool(s)/service(s), the author(s) reviewed and edited the content as needed and take(s) full responsibility for the publication's content.

References

[1] L. Wang, C. Ma, X. Feng, Z. Zhang, H. Yang, J. Zhang, Z. Chen, J. Tang, X. Chen, Y. Lin, et al., A survey on large language model based autonomous agents, Frontiers of Computer Science 18 (2024) 186345.

[2] J. C. Paiva, J. P. Leal, Á. Figueira, Automated assessment in computer science education: A state-of-the-art review, ACM Transactions on Computing Education (TOCE) 22 (2022) 1–40.

[3] S. Sarsa, P. Denny, A. Hellas, J. Leinonen, 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, 2022, pp. 27–43.

[4] B. A. Becker, P. Denny, J. Finnie-Ansley, A. Luxton-Reilly, J. Prather, E. A. Santos, Programming is hard-or at least it used to be: Educational opportunities and challenges of ai code generation, in: Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1, 2023, pp. 500–506.

[5] H. Tian, W. Lu, T. O. Li, X. Tang, S.-C. Cheung, J. Klein, T. F. Bissyandé, Is chatgpt the ultimate programming assistant–how far is it?, arXiv preprint arXiv:2304.11938 (2023).

[6] I. Estévez-Ayres, P. Callejo, M. Á. Hombrados-Herrera, C. Alario-Hoyos, C. Delgado Kloos, Evaluation of llm tools for feedback generation in a course on concurrent programming, International Journal of Artificial Intelligence in Education (2024) 1–17.

[7] D. Bengtsson, A. Kaliff, Assessment accuracy of a large language model on programming assignments, 2023.

[8] M. Messer, N. C. Brown, M. Kölling, M. Shi, Automated grading and feedback tools for programming education: A systematic review, ACM Transactions on Computing Education 24 (2024) 1–43.

[9] H. Gabbay, A. Cohen, Combining llm-generated and test-based feedback in a mooc for programming, in: Proceedings of the Eleventh ACM Conference on Learning@ Scale, 2024, pp. 177–187.

[10] Z. Xu, S. Jain, M. Kankanhalli, Hallucination is inevitable: An innate limitation of large language models, arXiv preprint arXiv:2401.11817 (2024).

[11] J. Li, S. Consul, E. Zhou, J. Wong, N. Farooqui, Y. Ye, N. Manohar, Z. Wei, T. Wu, B. Echols, et al., Banishing llm hallucinations requires rethinking generalization, arXiv preprint arXiv:2406.17642 (2024).

[12] B. Jiang, Y. Xie, Z. Hao, X. Wang, T. Mallick, W. J. Su, C. J. Taylor, D. Roth, A peek into token bias: Large language models are not yet genuine reasoners, arXiv preprint arXiv:2406.11050 (2024).

[13] N. Alzahrani, H. A. Alyahya, Y. Alnumay, S. Alrashed, S. Alsubaie, Y. Almushaykeh, F. Mirza, N. Alotaibi, N. Altwairesh, A. Alowisheq, et al., When benchmarks are targets: Revealing the sensitivity of large language model leaderboards, arXiv preprint arXiv:2402.01781 (2024).

[14] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, et al., Language models are few-shot learners, Advances in neural information processing systems 33 (2020) 1877–1901.

[15] U. Lee, Y. Kim, S. Lee, J. Park, J. Mun, E. Lee, H. Kim, C. Lim, Y. J. Yoo, Can we use gpt-4 as a

mathematics evaluator in education?: Exploring the efficacy and limitation of llm-based automatic assessment system for open-ended mathematics question, International Journal of Artificial Intelligence in Education (2024) 1–37.

[16] Y. Zhu, J. Li, G. Li, Y. Zhao, J. Li, Z. Jin, H. Mei, Hot or cold? adaptive temperature sampling for code generation with large language models, Proceedings of the AAAI Conference on Artificial Intelligence 38 (2024) 437–445. URL: https://ojs.aaai.org/index.php/AAAI/article/view/27798. doi:10.1609/aaai.v38i1.27798.

[17] M. Renze, E. Guven, The effect of sampling temperature on problem solving in large language models, arXiv preprint arXiv:2402.05201 (2024).

[18] S. Hong, C. Cai, S. Du, H. Feng, S. Liu, X. Fan, " my grade is wrong!": A contestable ai framework for interactive feedback in evaluating student essays, arXiv preprint arXiv:2409.07453 (2024).

[19] C.-M. Chan, W. Chen, Y. Su, J. Yu, W. Xue, S. Zhang, J. Fu, Z. Liu, Chateval: Towards better llm-based evaluators through multi-agent debate, arXiv preprint arXiv:2308.07201 (2023).

[20] W. McGugan, Beginning game development with Python and Pygame: from novice to professional, Apress, 2007.

[21] M.-J. Laakso, E. Kaila, T. Rajala, Ville–collaborative education tool: Designing and utilizing an exercise-based learning environment, Education and Information Technologies 23 (2018) 1655–1676.

[22] B. Johnson, Visual Studio Code: End-to-End Editing and Debugging Tools for Web Developers, John Wiley & Sons, 2019.

[23] D. Goodger, G. van Rossum, Docstring conventions, URL http://www. python. org/dev/peps/pep-0257 (2001).

[24] O. Topsakal, T. C. Akinci, Creating large language model applications utilizing langchain: A primer on developing llm apps fast, in: International Conference on Applied Engineering and Natural Sciences, volume 1, 2023, pp. 1050–1056.

[25] OpenAI, Pricing, 2025. URL: https://openai.com/api/pricing/.

[26] B. Jeffries, J. A. Lee, I. Koprinska, 115 ways not to say hello, world! syntax errors observed in a large-scale online cs0 python course, in: Proceedings of the 27th ACM Conference on on Innovation and Technology in Computer Science Education Vol. 1, ITiCSE '22, Association for Computing Machinery, New York, NY, USA, 2022, p. 337–343. URL: https://doi.org/10.1145/3502718.3524809. doi:10.1145/3502718.3524809.

[27] P. Organisciak, S. Acar, D. Dumas, K. Berthiaume, Beyond semantic distance: Automated scoring of divergent thinking greatly improves with large language models, Thinking Skills and Creativity 49 (2023) 101356.

A. Prompts

The prompts used in the autonomous agent pipeline.

A.1. Reviewer

System

You are a computer science university teacher teaching a fundamentals of programming course. Below is an exercise description, a rubric, and a student submission. Your task:

1. Provide detailed feedback on the submission using the rubric.
2. Give final marks at the end.

Output the final marks inside `<FINAL_MARKS>` tags. Please remember that all of the curly braces have been replaced with double curly braces by your instruction and students should not receive any penalty from double curly braces in their submission.

Human

You are a computer science university teacher teaching a fundamentals of programming course. Below is an exercise description, a rubric, and a student submission. Your task:

1. Provide detailed feedback on the submission using the rubric.
2. Give final marks at the end.

Output the final marks inside `<FINAL_MARKS>` tags. Please remember that all of the curly braces have been replaced with double curly braces by your instruction and students should not receive any penalty from double curly braces in their submission.

Exercise Description:
{exercise_description}

Rubric for evaluation:
{rubric}

Student submission:
{student_submission}

Please provide your assessment.

A.2. Commenter

System

You are a computer science university teacher teaching a fundamentals of programming course. You are part of a grade review board. Your task is to evaluate the validity and correctness of the previous teacher's grading based on the exercise, rubric, and the teacher's review. Then, provide your own final marks. Finally, output the final marks inside `<FINAL_MARKS>` tags. Please remember that all of the curly braces have been replaced with double curly braces by your instruction and students should not receive any penalty from double curly braces in their submission.

Human

Above are the original exercise description, rubric, and student submission (for your reference), plus the prior grading. Please provide your commentary and updated marks if needed.

A.3. Chairperson

System

You are the chairperson of the grade review board in a fundamentals of programming course. Your task:

1. Briefly justify the final decision.
2. Output the final marks inside `<FINAL_MARKS>` tags.

Please remember that all of the curly braces have been replaced with double curly braces by your instruction and students should not receive any penalty from double curly braces in their submission.

Human

Above is all relevant information:

- The original exercise description, rubric, and submission.
- The Reviewer and Commenter assessments.

Please provide the final marks as Chairperson. Remember to place them in `<FINAL_MARKS>` tags.