Int J Artif Intell Educ (2025) 35:774–790
https://doi.org/10.1007/s40593-024-00406-0

ARTICLE

Evaluation of LLM Tools for Feedback Generation in a Course on Concurrent Programming

Iria Estévez-Ayres[1] · Patricia Callejo[1] · Miguel Ángel Hombrados-Herrera[1] · Carlos Alario-Hoyos[1] · Carlos Delgado Kloos[1]

Accepted: 13 April 2024 / Published online: 15 May 2024

Abstract

The emergence of Large Language Models (LLMs) has marked a significant change in education. The appearance of these LLMs and their associated chatbots has yielded several advantages for both students and educators, including their use as teaching assistants for content creation or summarisation. This paper aims to evaluate the capacity of LLMs chatbots to provide feedback on student exercises in a university programming course. The complexity of the programming topic in this study (concurrency) makes the need for feedback to students even more important. The authors conducted an assessment of exercises submitted by students. Then, ChatGPT (from OpenAI) and Bard (from Google) were employed to evaluate each exercise, looking for typical concurrency errors, such as starvation, deadlocks, or race conditions. Compared to the ground-truth evaluations performed by expert teachers, it is possible to conclude that none of these two tools can accurately assess the exercises despite the generally positive reception of LLMs within the educational sector. All attempts result in an accuracy rate of 50%, meaning that both tools have limitations in their ability to evaluate these particular exercises effectively, specifically finding typical concurrency errors.

Keywords Generative AI · ChatGPT · Bard · Programming · Concurrency

✉ Iria Estévez-Ayres
ayres@it.uc3m.es

Patricia Callejo
pcallejo@it.uc3m.es

Miguel Ángel Hombrados-Herrera
mhombrad@it.uc3m.es

Carlos Alario-Hoyos
calario@it.uc3m.es

Carlos Delgado Kloos
cdk@it.uc3m.es

[1] Department of Telematic Engineering, Universidad Carlos III de Madrid, Avda. de la Universidad, 30, Leganés 28911, Madrid, Spain

Introduction

Self-regulated learning (SRL) is a style of learning that promotes students' awareness about their own beliefs, knowledge, motivations and cognitive processes (Butler & Winne, 1995). Self-regulated students can set learning goals, plan strategies to achieve them and adapt either their strategies or goals to manage their motivation and continue making progress. It has been well documented that students who engage in SRL tend to be more successful learners (Stehle & Peters-Burton, 2019; Butler & Winne, 1995).

The student deliberately monitors the learning process, asses it, and adapts in a closed loop. Precisely one of the distinctive features of SRL is a "self-oriented feedback loop" in the learning process (Carver & Scheier, 2012). This loop implies a cyclic continuous process in which students evaluate how fruitful is the outcome of their learning and act in consequence, either by adjusting their perception or by adjusting their learning strategies, engaging in behaviours that are more likely to improve exam performance, while avoiding behaviours that may not benefit success (Zimmerman, 1990).

Feedback has been proven to improve students learning in general. But in particular, feedback is a crucial catalyst for SRL (Butler & Winne, 1995). Students engaged in SRL obtain some internal feedback from monitoring their learning processes. Such feedback constitutes a measure of the quality of their cognitive processes and learning strategy. However, self-regulated learners can find discrepancies between the expected and the actual outcome of their activity and tend to draw upon external feedback, such as answered textbooks' questions, students' comments in collaborative groups, or instructors' remarks to address such disparity (Butler & Winne, 1995; Chou & Zou, 2020). A general consensus is that learners are more effective when using external feedback (Meyer, 1986; Hattie & Timperley, 2007).

From a different perspective, external feedback is even more relevant in recent years due to the extraordinary growth of online teaching, where instructors and students do not necessarily share the exact same spatial and time coordinates. Thus, it is even more pressing for the instructor to provide high-quality feedback to assist the students (Nicol & Macfarlane-Dick, 2006). Even more, in the last two decades, the advent and popularisation of learning management systems (LMS) has provided instructors with various methods to interact with students and content, and thus an unlimited opportunity for providing feedback (Cavalcanti et al., 2021).

Providing feedback for every student can be costly and time-consuming. It is hardly scalable when the feedback cannot be systematised (Dai et al., 2023). This is especially true when the instructor is in charge of teaching a highly specialised or complex matter. On the one hand, there are less resources available for the instructor. On the other hand, it becomes even more complicated for instructors to provide customised feedback to their students (Tarek et al., 2022). Here, customised is used in the sense of adapted to the student's output rather than in the understanding of adapted to the student's characteristics, as it is sometimes used in the literature. For feedback to be truly valuable, it has to be adapted to the student's output. As reported by Sadler (1989), good feedback must inform about a learning task or process to bridge the gap between the desired and the actual learning outcome.

Automating the feedback does not necessarily have to yield a worse performance than more traditional feedback. In fact, authors in Cavalcanti et al. (2021) performed a systematic literature review on automatic feedback that demonstrated that in 65.07% of studies reviewed, automatic feedback improved students' performance.

A popular field studied in recent years to provide personalised automatic feedback has been Natural Language Processing (NLP), which is an area of Artificial Intelligence (AI) focused on developing computational models able to understand and generate human language (Keuning et al., 2018).

The emergence of Large language models (LLM) at the end of 2022 has represented a drastic improvement in the expectations of NLP. LLMs are generative pre-trained models based upon the celebrated transformers (Vaswani et al., 2017), a deep learning model trained with vast amounts of textual data to learn language structures. This enables LLMs to perform tasks such as summarising texts, language translation or answering. Beyond that, LLMs represent a disruptive contribution with the so-called *emergent abilities*, the emergence of non-predicted capabilities in a larger model that was not designed or trained for it and that was not present in a smaller size model. Performing arithmetic operations, generating executable coding from natural language instructions, auto-debugging, or chain-of-thought prompting are only a few examples of reported emergent capabilities (Wei et al., 2022). A thorough benchmarking of these properties is being carried out within the AI community (Wei et al., 2022). This and other new properties, as well as potential pitfalls and limitations of the LLMs, are still a subject of study, and the plausibility of their use in different applications is yet to be confirmed. This search has not been exempt from controversy since the measure of these properties is complex (Schaeffer et al., 2023).

The latest versions of LLMs were trained with millions of public repositories of code, considerably improving their program capabilities. For this reason, there is an increasing interest in using these tools as code assistants. More specifically, using for this goal conversational agents (chatbots), that utilise a specific LLM architecture to provide human-like text-based conversations.

Learning how to program may be complex (Jenkins, 2002). In this process, effective feedback is a critical tool that can be used to improve student's performance and engagement (Marwan et al., 2020). After mastering programming fundamentals, the next step for most undergraduate engineering students is to study concurrent programming.

Concurrent programming allows programmers to improve software performance, taking advantage of multi-core systems (Wu et al., 2016). Concurrent programming also facilitates the implementation of complex systems using separation of concerns (Pinto et al., 2015). Depending on the platform and language, there are different units of execution (processes, threads, etc.). These units of execution, from now on *tasks*, share resources and may overlap during their execution. The programmer must guarantee that this overlapping of tasks is timed appropriately, thus avoiding any faulty behaviour.

Learning concurrent programming is challenging. First of all, students should change their mindsets from sequential to concurrent thinking, a huge paradigm shift (Sutter et al., 2005) that poses challenges even for more advanced programmers (Yu et al., 2017). Learners should avoid common pitfalls, such as *race conditions*

when concurrently accessing shared resources or defective communication patterns causing *deadlocks* or *starvation* (Hundt et al., 2017). Furthermore, the well-known non-deterministic nature of concurrent programs creates difficulties when attempting to replicate existing bugs during debugging sessions (Lu et al., 2008). In this setting, feedback becomes even more important due to the complexity of the subject (Hundt et al., 2017).

From the teachers' side, assessing concurrent programming manually can be challenging due to the simultaneous or interleaved execution of two or more processes. Thus, it is necessary to consider multiple execution flows when assessing the code (Paiva et al., 2022). However, due to their non-deterministic nature, automatic testing of concurrent programs is not a trivial task (Ihantola et al., 2010). For instance, automatic grading based on unit tests, which is relatively easy and reliable for sequential programs, becomes much less reliable when testing concurrent programs. This is because bugs are often subtle and may become visible only after many executions (Barros et al., 2023). Thus, the existing research focused mainly on developing tools to provide feedback to students, specifically targeting concurrency bugs like deadlocks or race conditions (Ihantola et al., 2010). Despite the publication of many articles (Melo et al., 2015), there are very few publicly available tools that have active user support (Barros et al., 2023). Additionally, the tools available require prior installation, configuration, and/or manually annotation of students' code (Blackshear et al., 2018; Calcagno et al., 2015; Pugh & Ayewah, 2007).

When assessing a potentially large number of complex student assignments, such as concurrent programming tasks, it is important for the tool used to be as user-friendly as possible for the teacher. Recent studies have shown that generative AI tools can provide such an easy-to-use experience while also offering reliable and easily understandable feedback for sequential programming (Zhang et al., 2024; Yilmaz & Karaoglan Yilmaz, 2023). This article explores the potential use of generative AI tools to assess concurrent programming assignments.

In this context, the research question this research aims to address is:

- **RQ**: Are LLMs reliable enough to independently provide feedback about concurrency programming problems?

This question is framed in the context of task-driven and static-check feedback. Task-driven feedback means feedback adapted to students's answers for problems without unique solutions (Deeva et al., 2021). Static check refers to feedback provided without compiling or running the code (Tarek et al., 2022) (i.e., programming style, program faults or design mistakes).

This paper is organised as follows: Section "Related Work" presents a review of related work in the field of automatic feedback, particularly in the context of Large Language Models (LLMs). Section "Methods" provides the details of the experiments conducted to address the research question of this paper, thanks to two LLMs chatbots that use LLMs (ChatGPT and Bard). Section "Results" describes the results. Section

"Discussion" discusses the outcomes of the experiments. Finally, Section "Limitations to This Study" concludes the paper.

Related Work

Researchers have extensively investigated the utilisation of automatic feedback for programming assignments. There are works focused on providing specific feedback about topics such as test failures, compiler errors, style issues, performance issues and more. The approaches proposed use a wide variety of techniques, different levels of automation and application domain (Keuning et al., 2018). Some of these works can be found compiled in Ala-Mutka (2005); Douce et al. (2005); Rahman and Nordin (2007); Lajis et al. (2018); Ihantola et al. (2010); Keuning et al. (2018); Paiva et al. (2022); Messer et al. (2024). In recent years, the introduction of AI in automatic feedback has become more prevalent.

A recent example focused on providing feedback to students in programming courses with the assistance of AI models is Afzaal et al. (2021). Their authors proposed an AI-based approach to provide feedback for self-regulation in students. They trained an explainable Machine learning (ML) model that offers visual recommendations through a dashboard. This ML model could predict the student's performance and provide advice accordingly.

The recent advancements in pre-trained LLMs since November 2022 have encouraged some researchers to use them for providing feedback on complex tasks with good results (Dai et al., 2023; Li & Xing, 2021; MacNeil et al., 2022; Sarsa et al., 2022; Mizumoto & Eguchi, 2023). For example, in Mizumoto and Eguchi (2023), authors automatically used ChatGPT to score 12,100 English essays for the TOEFL exam. They concluded that the GPT 3.5 model provided numerical scoring with enough accuracy and reliability to provide valuable support for human evaluations. Beyond a number score or grade, using LLMs for providing textual feedback has also been tested. In Dai et al. (2023), ChatGPT was used to provide textual feedback about data science project plans written by postgraduate students in a course about introductory data science. The model feedback was compared to instructors' feedback by a third-party pair of experts. They showed how GPT performed better than the instructors in readability as defined in Jia et al. (2022)). Nevertheless, the authors also demonstrated that GPT could not offer a valid assessment of student's performance compared to the instructor. The quality of the feedback was also evaluated, analysing the existence of effective feedback components with the theoretical framework proposed by Hattie and Timperley (2007). They reported how ChatGPT could effectively provide feedback about the student's learning process.

A similar approach was considered for programming courses. Sarsa et al. (2022) used OpenAI Codex with two purposes: to generate new programming exercises with solutions and to provide line-by-line explanations of student code. They reported that the explanations generated by the model were correct in 67.2% of the lines commented. Most erroneous lines contained only minor mistakes that the instructor could quickly correct.

To the best of our knowledge, up to the date of this paper's writing, no previous works evaluate the impact of automatic textual feedback generated assisted with generative language models oriented to provide feedback about concurrent programming.

Methods

This section describes the experiments conducted to answer the research question proposed in this paper. This work quantitatively analyses the LLMs chatbots' ability to identify typical synchronisation errors in college programming assessments, specifically ChatGPT and Bard.

Experiments

We conducted a total of four experiments that followed the same experimental design. The differences in the experiments depended on two variables: the chatbot used and the type of prompt sent to the chatbot (Table 1). The conversational agents used are ChatGPT (using GPT 3.5 as LLM) and Bard (with PaLM as LLM), which are the most popular chatbots, and both offer a free version.

In one of the experiments, labelled as "*basic*", the prompt consisted of plain instructions to the chatbot to detect synchronisation errors. When the prompt included synchronisation errors as an example, the experiment is labelled as "*with context*". The experiment labelled as "*with statement*" implies that the prompts additionally included the original statement handed to the students to perform the task. Further details about these are given in Section "Procedures".

Data

This research is supported by students' real answers about concurrent programming problems from a course on Systems Architecture. This course is taught in the fall semester of the second year of the Bachelor's Degree in Engineering at a Spanish university. Java is the programming language used in this course..

Table 1 Description of experiments conducted

	Experiment	Description
1	Bard basic	Ask Bard if an example is correct and detect any synchronisation problem
2	Bard w/ context	Ask Bard the same but previously providing it with some examples of synchronisation problems.
3	ChatGPT w/context	Same as 2, but with ChatGPT.
4	ChatGPT w/ context & w/ statement	Same as 3, but additionally providing to the LLM the original statement handed to the students.

The Java language is especially rich in concurrent programming constructs (Pinto et al., 2015). From all the possible synchronisation mechanisms of the language, the course focuses on two: monitors, a low-level mechanism supporting mutual exclusion and condition-based synchronisation and Semaphores, part of `java.util.concurrent`, a high-level library (Lea, 2005).

Students are asked to implement a correct concurrent program. Moreover, they are advised to pay special attention to avoiding race conditions (focusing on data races), deadlocks and starvation. *Race conditions* result when accesses to shared memory are not properly synchronised (Netzer & Miller, 1992). A *deadlock* happens when two or more operations circularly wait for each other to release the acquired resource (Holt, 1972), while in a starvation-free concurrent system, every process will eventually make progress (Schellhorn et al., 2016). All the answers were anonymous, every reference to students' identity was removed, and only the code was used as data.

More precisely, students were grouped in pairs (52 pairs), and they were asked to solve two different statements. They could choose if they solved them with *monitors* (34) or *semaphores* (18).

The extension of the answers is limited by the maximum number of tokens the language model accepts. For this reason, we pruned the code provided by students, eliminating comments and unnecessary spaces and shortening the variable names. As mentioned above, the problems' statements have also been included as input data in some variants of our experiments. The submissions present synchronisation errors of three types: Starvation, Deadlocks and Race conditions.

The course instructor thoroughly graded the exercises, and these corrections are used as the ground-truth for the experiments of this paper. Out of the 52 submissions, 10 (19%) presented Deadlocks, 11 (21%) presented Race conditions, 2 (4%) presented Starvation, and 35 (67%) were error-free.

Procedures

All the experiments conducted consisted of two stages (see Fig. 1): the first was for generating automatic feedback on all the available answers using the two considered chatbots under study, ChatGPT and Bard. In the second phase, the authors of this paper evaluated the feedback obtained from the chatbots.

For the generation phase, a *Python* script was implemented to prompt a language model for every submission. In the case of ChatGPT, the OpenAI API [1] was used, and in the case of Bard, a specific code was created to retrieve the responses. The input prompts consisted of the student's code in Java preceding the following basic instruction:

> *"Given the stated exercise. Is the syntax of this code correct? Do you identify any potential race condition, deadlock or starvation problem? Answer with the level of a university student, learning for the first time to program in Java, provide a maximum of 400 words response."*

[1] https://openai.com/blog/openai-api

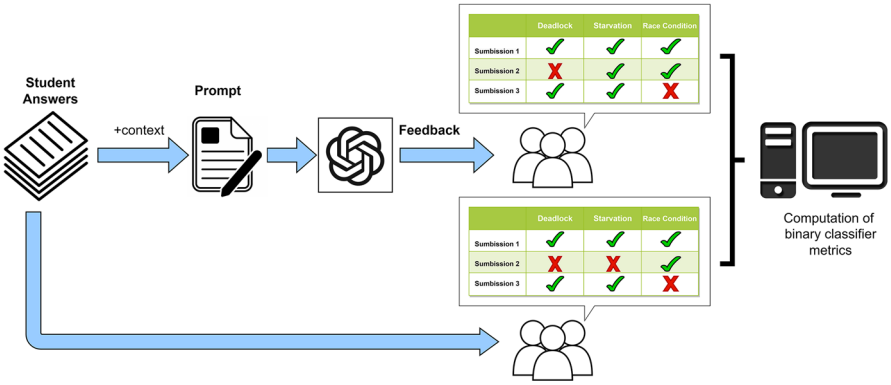

Fig. 1 Experiment setting

The *Python* script reads the student submission file, appends the instruction above and creates a prompt for the language model. Then, it collects and stores the model's feedback into a file. In the case of Bard, in most of cases, return three different responses at a time.

In order to exploit the context-based learning of the LLMs, commonly referred to as prompt engineering (Gao, 2023), we pushed forward subsequent versions of the experiment with more elaborated prompts to provide better context for the model. In the next iteration (Experiment 2), we extended the prompts with three example code fragment in Java that contain typical synchronisation errors (Deadlock, Race condition, and Starvation). One fragment per error was used. The three same examples are used in all the prompts. This setting was repeated with Bard and ChatGPT (Experiments 2 and 3).

Experiment 4 included, as an input prompt, on top of the three example code fragment, the original statement that was handed to the students as instruction for the problem.

The experiments were conducted with two different chatbots: Google *Bard* as of June 2023, using PaLM 2 as LLM; and ChatGPT, using *GPT 3.5 turbo 16k* as of July 2023 as LLM, configured with a temperature parameter of 0.8.

During the evaluation phase, three teaching staff members from an engineering department were involved in evaluating the data and results. An instructor checked each student's submission to detect the presence or absence of any of the three synchronisation errors. This procedure yielded a table of binary "yes/no" values that constitute the ground-truth for our experiments. Additionally, the chatbots' feedback was checked on each submission to verify if the results matched the ground-truth. In this case, two different teaching staff members checked each model's feedback twice to identify errors and spot any lack of self-consistency in the answers. For example, the model might confirm the presence of an error, and yet, within the same response in the following paragraphs, it adds information that conflicts with the first statement.

Metrics

The results obtained result in binary tables “yes/no" stating whether the error is present or not. For each experiment and error, a confusion matrix has been computed. These are evaluated using the well-known performance metrics for binary variables: accuracy, precision, recall, and F1-score (Fielding & Bell, 1997). For computing them, “True Positive" is the number of cases in which the model confirmed an existing error. “True Negative" is the number of cases in which the model omitted an error not present in the submission. “False Positive" is the number of times the model incorrectly confirmed the presence of a mistake. Finally “False Negative" is the number of times the model missed an error in the submission.

Results

On each experiment, the output provided by the LLMs chatbots consisted of about 52 feedback in the form of text, each corresponding to a student submission. As described above, each feedback was thoroughly evaluated by experts to determine whether it indicated any of the synchronisation errors (deadlock, starvation, or race condition).

By default, Bard provides in most cases three alternative answers for each prompt. Due to the probabilistic nature of the language models, on a preliminary test we verified that the three answers provided for each answer were very similar in all the cases, not contradicting each other nor changing substantially the information provided in most of the cases. Therefore, the results are based on only one of the three instances.

The answers provided by the chatbots consisted of a text of several paragraphs that followed a similar structure in most of the cases:

- An introductory statement about the correctness of the code syntax and the presence of any of the three synchronisation errors. The latter is almost always stated in tentative language, using expressions such as “not obvious" or “potential errors". The opening statements rarely assert the presence of an error; it is just suggested. Examples of this are displayed in the Table 2.
- Then, a generic brief description of the three types of errors: deadlock, starvation and race condition. The definitions are correct but not practical as task-driven feedback, as it is the same for all the submissions.

Table 2 Example of opening statements provided by an LLM

ChatGPT	“*The syntax of the provided code appears to be correct. However, there are potential Race conditions, Deadlocks, and Starvation problems that need to be addressed.*"
Bard	“*The syntax of the code is correct. However, there are a few potential Race conditions, Deadlocks, or Starvation problems.*"

- A more detailed explanation about the presence and location of the errors. This can be considered task-driven feedback in the sense that it provides an explanation of the actual error occurring in the code provided, and for that, the chatbot uses correct names for the variables, methods and classes defined by the student (i.e: Table 3). However, in many cases the logic of the explanation is incorrect (see Table 3). Furthermore, in some cases, this explanation contradicts the suggestion made in the opening statement.

Table 3 Examples of feedback where the model correctly spotted and described Deadlocks and Race conditions in the submission

Deadlock	Bard w/context	"***Deadlock:** There is a potential deadlock in the 'elevadorOutTur()' and 'elevadorOutMago()' methods. This is because the threads that are waiting to exit the elevator must wait for the elevator to be empty, and the elevator can only be empty when there are no more people waiting to ride. This could lead to a situation where the threads are stuck waiting for each other, and neither can proceed.*"
Deadlock	ChatGPT w/statement	"*In the code, there is a potential deadlock situation: If a thread is waiting for another thread to release the 'alqsSala' variable, and the other thread is waiting for the first thread to release the 'alqsPasillo' variable, then the two threads will be deadlocked.* "
Race condition	Bard	"***Race condition:** There is a race condition in the 'elevadorInTur()' and 'elevadorInMago()' methods. This is because multiple threads could be trying to enter the elevator at the same time. This could lead to the wrong person being chosen to ride the elevator, or to the elevator being full even though there are still people waiting to ride.*"
Race condition	ChatGPT w/statement	*"Similarly, the methods 'elevadorInTur()' and 'elevadorInMago()' are synchronized, but they use a 'while' loop to check if there is space in the elevator. However, the 'elevadorLibre' variable is not updated within the loop, which means that even if a thread is waiting, it may still enter the elevator if another thread has already entered. This can cause a race condition where more than the allowed number of threads enter the elevator. In addition, the 'toin()' and 'toout()' methods in the 'Elevador' class are synchronized, but they do not use a 'while' loop to check if the condition for execution is still true. This means that a thread may wake up from 'wait()' and proceed to execute the code, even if the condition has changed and it should not proceed. This can cause race conditions when threads try to access the 'numgente' variable."*

Table 4 Summary of the benchmarking

Starvation	Accuracy	Precision	Recall	F1 Score
Bard Basic	0.4	0.07	1.0	0.14
Bard w/ context	0.5	0.1	1.0	0.17
ChatGPT w/ context	0.6	0.0	0.0	-
ChatGPT w/ context & w/ statement	0.62	0.05	0.5	0.09

Comparison between the experts' assessment and the interpretation of the chatbot feedback about starvation errors

- In some instances, an extra paragraph is added with further information suggesting how to improve the efficiency or quality of the code, or remarks on an example of a good practice present in the answer, such as the use of conditional variables. In these cases, the feedback provides specific examples from the answer, and it names variables defined by the student. Besides that, in some examples the feedback provides general advice for avoiding synchronisation problems, as well as some general advice to improve programming practices, such as: *"The code is well-commented, which is helpful for a university student learning to program in Java."*.
- Generic closing statement or final remarks with a practical absence of any personalisation.

In order to quantify the reliability of the feedback, each one was inspected to determine if it identified an error or not. The feedback was labelled by the experts with "yes" if it states undoubtedly the presence of a particular error. Conversely, if the error type is not mentioned or just suggested tentatively, the feedback is labelled with "no".

For each experiment and error type, a confusion matrix was computed to calculate the binary classification metrics shown in Tables 4, 5 and 6

Discussion

The summary of results presented in the previous section (Tables 4, 5 and 6) show a low correlation between the instructor feedback and the feedback provided by the chatbots. The general trend of the chatbots is to be excessively penalising. Specifically,

Table 5 Summary of the benchmarking

Deadlock	Accuracy	Precision	Recall	F1 Score
Bard Basic	0.6	0.22	0.57	0.32
Bard w/ context	0.37	0.23	1.0	0.37
ChatGPT w/ context	0.48	0.13	0.3	0.18
ChatGPT w/ context & w/ statement	0.52	0.22	0.6	0.32

Comparison between the experts' assessment and the interpretation of the chatbot feedback about errors

Table 6 Summary of the benchmarking

Race condition	Accuracy	Precision	Recall	F1 Score
Bard Basic	0.74	0.44	0.89	0.59
Bard w/ context	0.32	0.23	0.78	0.35
ChatGPT w/ context	0.33	0.18	0.64	0.29
ChatGPT w/ context & w/ statement	0.4	0.26	1.0	0.42

Comparison between the experts' assessment and the interpretation of the chatbot feedback about Race condition errors

while recall values are close to the unit in many cases, precision values are consistently low, with the highest precision value 0.44 (Table 6). This implies that classification is overly permissible or risk-averse, as chatbots are prone to identify errors even when the evidence is weak or ambiguous. This is problematic in the application proposed as for a potential tool intended to assist professors in providing feedback. It is preferable to miss an error present on a student assignment than to indicate an error where there is none, leading the student to confusion.

The best performance was achieved in the first experiment with Bard for predicting race conditions. It produced an accuracy of 0.74 and an F1 score of 0.59 (Table 6). It is interesting to remark that whereas the addition of more context to the prompt consistently increased the reliability of the feedback when using ChatGPT, it had an obvious negative effect on Google Bard feedback (See Table 6). Namely, in some cases, adding context to Bard aggravated its performance. A potential explanation has to do with the *context window*'s size of the models, which is a parameter fixed by the model's architecture and determines the contextual understanding range of the inputs. That is to say, how many preceding tokens (words for the sake of the explanation) does the model consider to generate the prediction of the next token in a sequence (Radford et al., 2018). The version of the GPT model used in this work has a larger contextual window than. This can explain why Bard does worse, given the length of the inputs, as it cannot simultaneously capture the relevant information from the answer and the context appended. Therefore, giving a larger input can be counterproductive if the context window is not large enough, as the attention of the context window will be more frequently centred on information that is not relevant to provide a better answer. And there is no guarantee that the valuable information to identify a synchronisation error is concentrated in a few consecutive lines of code but somewhat scattered throughout the whole code.

Regarding the type of error, three out of four experiments showed that LLMs performed better at identifying race conditions. Results are considerably inferior for starvation errors concerning the other. This can be explained by the fact that the total number of starvation errors only amounts to two, making them models even more likely to yield false positives.

As mentioned earlier, a concurrent program is challenging to debug as problems like race conditions, deadlocks, or starvation are difficult to replicate in an interactive session. In fact, even after fixing a project, some patches released by programmers are still buggy (Lu et al., 2008). We believe that one of the problems is not the lack of

concurrent code used for training the LLMs but how they use (and misuse) the different synchronisation mechanisms provided by the language and platforms. Various studies (Pinto et al., 2015; Wu et al., 2016) analysing public repositories of code found that 75% of Java projects and $\approx$ 10% of C# projects were concurrent programs. This means they divided their execution into tasks, and more than half of these projects used locks and mutexes for synchronisation. However, the same studies found that using advanced synchronisation mechanisms such as semaphores or monitors in Java or promises and futures in C# was not widely used and, in some cases, even misused. Therefore, if LLMs rely on the correctness of the code used for training and this code is faulty, the feedback would be unreliable and insufficient for an educational setting where high-quality feedback with good accuracy is paramount.

Limitations to this Study

The experiment presented in this study was undertaken using ChatGPT 3.5 and Bard with PaLM 2 due to budget reasons. Furthermore, the limited number of students' assignments can be considered another limitation.

The authors recognize that this field is constantly evolving, with new solutions emerging every week that improve upon previous ones. In the last two years, the accelerated evolution of the field has become evident. Companies such as OpenAI and Google have released new and improved versions of their LLMs with more parameters and new capabilities. For instance, in 2021 alone, OpenAI released seven updates of GPT 3.5 turbo. The authors acknowledge that the results obtained in this work may soon become outdated due to this fast-paced development. However, they can still be useful as a valuable reference or benchmark for future work.

The authors also acknowledge that techniques such as fine-tuning (Church et al., 2021; Radiya-Dixit & Wang, 2020) or RAG (Retrieval Augmented Generation) (Chen et al., 2024) could improve the results obtained. However, the objective of this article is to assess the capabilities of providing automatic feedback using GenAI solutions using prompt engineering applied to concurrent programming, but not to compare and improve existing LLMs.

Another limitation of this study is the number of available exercises. The data used in this study come from code assignments of students enrolled in a concurrent programming course on Systems Architecture. As the 104 students were organized in pairs, the number of submissions that could be used in the study were only 52. Although the number of exercises can be a limitation, we believe that using the code of a course edition should be sufficient to see the impact that this approach can have on education.

Conclusions and Future Work

The evolution of large language models (LLMs) has just begun; consequently, there is an enormous potential for improvement in the coming future. Their success as code

assistants, like GitHub Copilot, is garnering interest in introducing domain-specific knowledge.

In this paper, we explored to what extent large language models can be useful in assisting in concurrent programming tasks. In particular, we formulated the research question: *"Are LLMs reliable enough to independently provide feedback about concurrency programming problems?"*. Despite the reported success of the LLMs in assisting with programming tasks in general, our experimental results using ChatGPT 3.5 and Bard with PaLM 2 showed that they are not yet reliable for providing feedback on concurrency programming. However, with the evolution of technology and the continuous development of LLMs, it is possible that LLMs specifically trained in this area or newer versions of evaluated LLMs could yield better results.

Author Contributions IEA conceptualized the study and realized data collection. IEA, PC and MAHH designed the work and analyzed the data. All authors contributed to the interpretation of data. CDK and CAH supervised the study. MAHH, PC and IEA wrote the manuscript draft. CDK, CAH, PC and MAHH provided the funding. All authors revised the draft and contributed to the final manuscript.

Funding Open Access funding provided thanks to the CRUE-CSIC agreement with Springer Nature. This work was supported in part by grant PID2020-112584RB-C31 (H2O Learn project) funded by MCIN/AEI/10.13039/501100011033, by grant PID2022-141045OB-C43 (Generation of Reliable Synthetic Health Data for Federated Learning in Secure Data Spaces)) funded by the Spanish Ministry of Science, Innovation and Universities, and in part by the Madrid Regional Government through the Multiannual Agreement with UC3M in the line of Excellence of University Professors EPUC3M21, and in the context of the V PRICIT (Regional Programme of Research and Technological Innovation), a project which is co-funded by the European Structural Funds (FSE and FEDER). Partial support has also been received from the European Commission through the Erasmus+ projects MICROCASA (101081924-ERASMUS-EDU-2022-CBHE-STRAND-2), EUCare4.0 (2021-1-FR01-KA220-VET-000024860), and POEM-SET (2021-FR01-KA220-HED-000032171). Funding for APC: Universidad Carlos III de Madrid (Agreement CRUE-Madroño 2024). This publication reflects the views only of the authors, and funders cannot be held responsible for any use which may be made of the information contained therein.

Data Availability No datasets were generated or analysed during the current study.

Declarations

Competing Interests The authors declare no competing interests.

References

Afzaal, M., Nouri, J., Zia, A., Papapetrou, P., Fors, U., Wu, Y., Li, X., & Weegar, R. (2021). Explainable ai for data-driven feedback and intelligent action recommendations to support students self-regulation. *Frontiers in Artificial Intelligence., 4*, 723447.

Ala-Mutka, K. M. (2005). A survey of automated assessment approaches for programming assignments. *Computer Science Education, 15*(2), 83–102.

Barros, M., Ramos, M., Gomes, A., Cunha, A., Pereira, J., & Almeida, P. S. (2023). An experimental evaluation of tools for grading concurrent programming exercises. In M. Huisman & A. Ravara (Eds.), *Formal Techniques for Distributed Objects, Components, and Systems* (pp. 3–20). Cham: Springer.

Blackshear, S., Gorogiannis, N., O'Hearn, P. W., & Sergey, I. (2018). Racerd: compositional static race detection. *Proceedings of the ACM on Programming Languages,2*(OOPSLA). https://doi.org/10.1145/3276514

Butler, D. L., & Winne, P. H. (1995). Feedback and self-regulated learning: A theoretical synthesis. *Review of Educational Research, 65*(3), 245–281.

Calcagno, C., Distefano, D., Dubreil, J., Gabi, D., Hooimeijer, P., Luca, M., O'Hearn, P., Papakonstantinou, I., Purbrick, J., & Rodriguez, D. (2015). Moving fast with software verification. In K. Havelund, G. Holzmann, & R. Joshi (Eds.), *NASA Formal Methods* (pp. 3–11). Cham: Springer.

Carver, C.S., & Scheier, M.F.: Attention and Self-regulation: A Control-theory Approach to Human Behavior. Springer (2012)

Cavalcanti, A. P., Barbosa, A., Carvalho, R., Freitas, F., Tsai, Y.-S., Gašević, D., & Mello, R. F. (2021). Automatic feedback in online learning environments: A systematic literature review. *Computers and Education: Artificial Intelligence., 2*, 100027.

Chen, J., Lin, H., Han, X., Sun, L. (2024) Benchmarking large language models in retrieval-augmented generation. In *Proceedings of the AAAI Conference on Artificial Intelligence* (vol. 38, pp. 17754–17762)

Chou, C.-Y., & Zou, N.-B. (2020). An analysis of internal and external feedback in self-regulated learning activities mediated by self-regulated learning tools and open learner models. *International Journal of Educational Technology in Higher Education, 17*(1), 1–27.

Church, K. W., Chen, Z., & Ma, Y. (2021). Emerging trends: A gentle introduction to fine-tuning. *Natural Language Engineering, 27*(6), 763–778. https://doi.org/10.1017/S1351324921000322

Dai, W., Lin, J., Jin, F., Li, T., Tsai, Y.-S., Gasevic, D., Chen, G. (2023). Can large language models provide feedback to students? a case study on chatgpt.

Deeva, G., Bogdanova, D., Serral, E., Snoeck, M., & De Weerdt, J. (2021). A review of automated feedback systems for learners: Classification framework, challenges and opportunities. *Computers & Education., 162*, 104094.

Douce, C., Livingstone, D., & Orwell, J. (2005). Automatic test-based assessment of programming: A review. *Journal on Educational Resources in Computing (JERIC)., 5*(3), 4.

Fielding, A. H., & Bell, J. F. (1997). A review of methods for the assessment of prediction errors in conservation presence/absence models. *Environmental Conservation, 24*(1), 38–49.

Gao, A. (2023). Prompt engineering for large language models. Available at SSRN 4504303.

Hattie, J., & Timperley, H. (2007). The power of feedback. *Review of Educational Research, 77*(1), 81–112.

Holt, R. C. (1972). Some deadlock properties of computer systems. *ACM Computing Surveys (CSUR)., 4*(3), 179–196.

Hundt, C., Schlarb, M., & Schmidt, B. (2017). Sauce: A web application for interactive teaching and learning of parallel programming. *Journal of Parallel and Distributed Computing., 105*, 163–173.

Ihantola, P., Ahoniemi, T., Karavirta, V., Seppälä, O. (2010). Review of recent systems for automatic assessment of programming assignments. In *Proceedings of the 10th Koli Calling International Conference on Computing Education Research* (pp. 86–93).

Jenkins, T. (2002). On the difficulty of learning to program. In *Proceedings of the 3rd Annual Conference of the LTSN Centre for Information and Computer Sciences* (vol. 4, pp. 53–58). Citeseer.

Jia, Q., Young, M., Xiao, Y., Cui, J., Liu, C., Rashid, P., Gehringer, E. (2022). Insta-reviewer: A data-driven approach for generating instant feedback on students' project reports. *International Educational Data Mining Society*.

Keuning, H., Jeuring, J., & Heeren, B. (2018). A systematic literature review of automated feedback generation for programming exercises. *ACM Transactions on Computing Education (TOCE)., 19*(1), 1–43.

Lajis, A., Baharudin, S. A., Ab Kadir, D., Ralim, N. M., Nasir, H. M., & Aziz, N. A. (2018). A review of techniques in automatic programming assessment for practical skill test. *Journal of Telecommunication, Electronic and Computer Engineering, 10*(2–5), 109–113.

Lea, D. (2005). The java. util. concurrent synchronizer framework. *Science of Computer Programming., 58*(3), 293–309.

Li, C., & Xing, W. (2021). Natural language generation using deep learning to support mooc learners. *International Journal of Artificial Intelligence in Education, 31*, 186–214.

Lu, S., Park, S., Seo, E., Zhou, Y. (2008) Learning from mistakes: a comprehensive study on real world concurrency bug characteristics. In *Proceedings of the 13th International Conference on Architectural Support for Programming Languages and Operating Systems* (pp. 329–339).

MacNeil, S., Tran, A., Mogil, D., Bernstein, S., Ross, E., Huang, Z. (2022). Generating diverse code explanations using the gpt-3 large language model. In *Proceedings of the 2022 ACM Conference on International Computing Education Research* (vol. 2, pp. 37–39).

Marwan, S., Gao, G., Fisk, S., Price, T.W., Barnes, T. (2020). Adaptive immediate feedback can improve novice programming engagement and intention to persist in computer science. In *Proceedings of the 2020 ACM Conference on International Computing Education Research* (pp. 194–203).

Melo, S.M., Souza, S.R.S., Silva, R.A., Souza, P.S.L. (2015). Concurrent software testing in practice: a catalog of tools. In *Proceedings of the 6th International Workshop on Automating Test Case Design, Selection and Evaluation. A-TEST 2015* (pp. 31–40). New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/2804322.2804328 .

Messer, M., Brown, N. C. C., Kölling, M., & Shi, M. (2024). Automated grading and feedback tools for programming education: A systematic review. *ACM Trans. Comput. Educ.,24*(1). https://doi.org/10.1145/3636515

Meyer, L. A. (1986). Strategies for correcting students' wrong responses. *The Elementary School Journal, 87*(2), 227–241.

Mizumoto, A., & Eguchi, M. (2023). Exploring the potential of using an ai language model for automated essay scoring. *Research Methods in Applied Linguistics., 2*(2), 100050.

Netzer, R. H., & Miller, B. P. (1992). What are race conditions? some issues and formalizations. *ACM Letters on Programming Languages and Systems (LOPLAS)., 1*(1), 74–88.

Nicol, D. J., & Macfarlane-Dick, D. (2006). Formative assessment and self-regulated learning: A model and seven principles of good feedback practice. *Studies in Higher Education, 31*(2), 199–218.

Paiva, J. C., Leal, J. P., & Figueira, A. (2022). Automated assessment in computer science education: A state-of-the-art review. *ACM Transactions on Computing Education (TOCE),22*(3). https://doi.org/10.1145/3513140

Pinto, G., Torres, W., Fernandes, B., Castor, F., & Barros, R. S. (2015). A large-scale study on the usage of java's concurrent programming constructs. *Journal of Systems and Software, 106*, 59–81.

Pugh, W., & Ayewah, N. (2007). Unit testing concurrent software. In *Proceedings of the 22nd IEEE/ACM International Conference on Automated Software Engineering. ASE '07* (pp. 513–516). New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/1321631.1321722 .

Radford, A., Narasimhan, K., Salimans, T., Sutskever, I., et al (2018). Improving language understanding by generative pre-training.

Radiya-Dixit, E., Wang, X. (2020). How fine can fine-tuning be? learning efficient language models. In Chiappa, S., Calandra, R. (eds.), *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics. Proceedings of Machine Learning Research* (vol. 108, pp. 2435–2443). PMLR. https://doi.org/10.48550/arXiv.2004.14129

Rahman, K.A., Nordin, M.J. (2007). A review on the static analysis approach in the automated programming assessment systems.

Sadler, D. R. (1989). Formative assessment and the design of instructional systems. *Instructional Science, 18*, 119–144.

Sarsa, S., Denny, P., Hellas, A., Leinonen, J. (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* (vol. 1, pp. 27–43).

Schaeffer, R., Miranda, B., Koyejo, S. (2023). Are emergent abilities of large language models a mirage? arXiv:2304.15004

Schellhorn, G., Travkin, O., Wehrheim, H. (2016). Towards a thread-local proof technique for starvation freedom. In *Integrated Formal Methods: 12th International Conference, IFM 2016, Reykjavik, Iceland, June 1-5, 2016, Proceedings 12* (pp. 193–209). Springer.

Stehle, S. M., & Peters-Burton, E. E. (2019). Developing student 21st century skills in selected exemplary inclusive stem high schools. *International Journal of STEM education., 6*(1), 1–15.

Sutter, H., et al. (2005). The free lunch is over: A fundamental turn toward concurrency in software. *Dr. Dobb's Journal., 30*(3), 202–210.

Tarek, M., Ashraf, A., Heidar, M., Eliwa, E. (2022) Review of programming assignments automated assessment systems. In: *2022 2nd International Mobile, Intelligent, and Ubiquitous Computing Conference (MIUCC)* (pp. 230–237). IEEE.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł, & Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems.,30*

Wei, J., Tay, Y., Bommasani, R., Raffel, C., Zoph, B., Borgeaud, S., Yogatama, D., Bosma, M., Zhou, D., Metzler, D., et al (2022). Emergent abilities of large language models. arXiv:2206.07682

Wu, D., Chen, L., Zhou, Y., & Xu, B. (2016). An extensive empirical study on c++ concurrency constructs. *Information and Software Technology, 76*, 1–18.

Yilmaz, R., & Karaoglan Yilmaz, F. G. (2023). The effect of generative artificial intelligence (ai)-based tool use on students' computational thinking skills, programming self-efficacy and motivation. *Computers and Education: Artificial Intelligence., 4*, 100147. https://doi.org/10.1016/j.caeai.2023.100147

Yu, F., Zhong, H., Shen, B. (2017). How do programmers maintain concurrent code? In *2017 24th Asia-Pacific Software Engineering Conference (APSEC)* (pp. 594–599). IEEE.

Zhang, Z., Dong, Z., Shi, Y., Price, T., Matsuda, N., Xu, D. (2024) Students' perceptions and preferences of generative artificial intelligence feedback for programming. In *Proceedings of the AAAI Conference on Artificial Intelligence* (vol. 38, pp. 23250–23258).

Zimmerman, B. J. (1990). Self-regulated learning and academic achievement: An overview. *Educational Psychologist., 25*(1), 3–17.