Leveraging GenAI for an Intelligent Tutoring System for R: A Quantitative Evaluation of Large Language Models

Lukas Frank*†, Fabian Herth*†, Paul Stuwe*†, Marco Klaiber, Felix Gerschner and Andreas Theissler
Aalen University of Applied Sciences, Aalen, Germany
*Corresponding authors: {fabian.herth, paul.stuwe, lukas.frank}@studmail.htw-aalen.de
†Authors contributed equally.

2024 IEEE Global Engineering Education Conference (EDUCON) | 979-8-3503-9402-3/24/$31.00 ©2024 IEEE | DOI: 10.1109/EDUCON60312.2024.10578933

***Abstract*—The tremendous advances in Artificial Intelligence (AI) open new opportunities for education, with Intelligent Tutoring Systems (ITS) powered by Generative Artificial Intelligence (GenAI) proving to be a promising prospect. Because of this, our work explores state-of-the-art (SOTA) ITS approaches with the integration of Large Language Models (LLMs) to improve programming education. We investigate whether and how a GenAI-based ITS can effectively support students in learning R programming skills. We measured the performance of three current pairings of LLMs and user interfaces: GPT-3.5 via ChatGPT, PaLM 2 via Google Bard, and GPT-4 via Bing. Therefore, we evaluated the LLMs on four types of problem settings when learning/teaching programming. Our experimental results show that the use of generative AI, specifically LLMs for R programming, is promising, where GPT-3.5 yielded the most satisfactory results. Furthermore, the advantages and limitations of our approach are addressed and revealed. Finally, open research directions towards explainable AI (XAI) and integrated self-assessment are pointed out.**

***Index Terms*—Generative AI, AI in Education, Intelligent Tutoring Systems, R Programming, Student Support**

I. Introduction

Programming courses are an integral part of contemporary higher education curricula, especially within STEM fields. However, prior programming knowledge and skills differ among students entering these courses, which presents an ongoing challenge: the discrepancy in skill levels exhibited vs. required may lead to a mismatch between course content and students' needs, potentially resulting in student dissatisfaction, disengagement, or suboptimal learning outcomes. At the same time, instructors and (human) tutors may not be able to adequately aid each individual student due to limited communication time between instructors and students [1], especially in light of Massive Open Online Courses (MOOCs) with class sizes in the thousands [2].

To bridge this gap, Intelligent Tutoring Systems (ITS) appear as a promising solution. Escotet [3] describes them as applications providing personalized instruction and student feedback, as well as exercises adapted to individual students' learning types. These approaches can improve student participation and academic performance [3]. An ITS allows students to progress at their own pace and adapts to the individual needs of the learner [2]. A recent study by Phung et al. [4] demonstrates the potential of ITS supported by Generative Artificial Intelligence (GenAI), acting as a personalized tutor for students or an assistant for instructors. Their approach has been shown to exhibit satisfactory performance within certain domains, compared to previous developments, and especially compared to human tutors [4]. Su and Yang [5] state that GenAI-powered Intelligent Tutoring Systems are able to supplement or even replace human instructors in certain cases. However, we found that existing studies on ITS performance primarily consider selected major programming languages (as indicated by the TIOBE index[1]), such as Python, Java, or C++. This presents an opportunity to conduct a study on the performance of ITS for less widely used programming languages, such as R, which is nevertheless an important component in contemporary data science curricula (see e.g. [6]–[8]).

The R programming language is an open source environment used for statistical computing and visualization [9]. On the 2022 IEEE spectrum of "top programming languages", it is ranked 8th out of 57 [10]. Considering its relevance and use for introductory programming courses in undergraduate courses on data science, statistics, and further topics, we argue that the use of LLMs as tutoring systems for the R programming language lends itself to evaluation. Compared to Python, the current leader of the TIOBE index, R is very similar, as shown by [11]. This similarity makes the evaluation of an ITS for R programming education all the more interesting, as it allows a more direct comparison between the performance of the ITS found in scientific literature.

In light of these considerations, the goal of this paper is to explore the performance of a GenAI-based ITS for the education of R programming. Building on the results outlined by Phung et al. [4], this study aims to highlight the opportunities (and challenges) posed by using AI-based tutoring for the R programming language. Specifically, we seek to address the question of whether a GenAI-based ITS serves as an effective tool to assist students in learning the R programming language.

Therefore, this paper makes the following contributions:

1) An analysis of whether LLMs are sufficiently powerful to guide (student) users in learning the R programming

[1]https://www.tiobe.com/tiobe-index/

language
2) Evaluation of whether LLMs can take on the role of a (human) tutor in the learning process
3) The comparison of three SOTA LLMs: GPT-3.5, GPT-4 and PaLM 2
4) Identifying research gaps and open topics for future work

This work is structured as follows: In Section II, an overview of the relevant topics within the scope of this paper is given. Related work is then presented in Section III, whereby we also delimit our own research. Section IV then describes the design and methodology of our experimental study on the evaluation of our ITS. The results are presented in Section V and discussed, especially with regard to the limitations of our study, in Section VI. Subsequently, our limitations and directions for future work are outlined in Section VII. Finally, the conclusion is presented in Section VIII.

II. Background

To obtain a comprehensive picture of the state of AI-supported programming education, a structured approach is undertaken considering the literature on which our work is based. First, the contemporary use of AI in education is described, specifically focusing on ITS. Additionally, an overview of the main attributes of the R programming language is given. Next, characteristics of human-based programming instruction and approaches for developer skill assessment are presented. Based on this, sufficient understanding for the ITS performance assessment is provided to the reader in the next chapter.

A. *Artificial Intelligence in Education*

In education, the inclusion of AI has permeated many areas and led to a variety of innovative use cases, such as in evaluation and assessment, AI-powered assistants, learning management for students, gamification, virtual environments, or new opportunities for collaborative learning [2], [12]. In addition, recent developments, explicitly in the area of LLMs, have increased AI performance (and application) [13]. Accordingly, this rise in popularity since the beginning of the last decade is also seen as an "AI boom" [14]. This increase in AI offers the opportunity for students to be placed even more at the centre of the learning experience, allowing teachers to spend more time with individual students [12]. Furthermore, AI in education can even be strategic and reduce the workload of teachers and students, especially in the face of a teacher shortage [15].

B. *Intelligent Tutoring Systems*

Intelligent Tutoring Systems bring the ability to target student knowledge and have emerged over the years as particularly suitable for programming due to the structured nature of computer programs [16]. There are studies, for example, by Troussas et al. [17] and Bey and Champagnat [1], which state that the use of an ITS improves pupils' performance in a statistically significant way. Early systems used to find bugs in programs written in the Pascal programming language, for example, produced satisfactory results. Consequently, ITS have continued to evolve and different generations have emerged, ranging from "expert systems" developed in the 1970s to modern AI-based systems that have been emerging since 2015 [18]. According to Zhai et al. [15], there are four ways in which an ITS provides a personalized learning experience:

1) Monitoring the student's input
2) Delivering appropriate tasks
3) Providing effective feedback
4) Applying an interface for human-computer communication.

Regarding the architecture of an ITS, several approaches can be found in the literature. For these purposes, we refer to the approach of Abu-Naser et al. [19], where the "classical ITS architecture" is described, which consists of the following components with an overarching coordination module. However, we point out that the role and functionality of the components may differ in the scientific community, but the following division is still popular within research:

1) Expert module: able to compile/parse arbitrary code
2) Pedagogical module: containing the course's subject matter
3) Student module: used for user management, including learning progress
4) User interface: encompassing a GUI
5) (optional) Problem generation module: generate code templates to be solved

C. *Programming instruction and developer skill*

To further the understanding of the requirements of the didactic approach for our ITS, the manner in which programming skills are classically taught to students needs to be understood. Brusilovsky et al. [20] enumerate three typical tutoring approaches, with the "sublanguage" approach, where only a small percentage of commands of the target language is taught, is described to exhibit the best results within programming courses. Furthermore, course success depends on avoiding cognitive overload, visualization techniques, embedding the material in relevant contexts, and the use of a good programming environment [20]. Although Brusilovsky's views are primarily applicable to procedural programming languages, they can also be applied to object-oriented programming (OOP), provided that the above-mentioned requirements are met, especially for programming environments with extended visualization capabilities [20]. Crestani and Sperber [21] note that conventional tutoring approaches (i.e. simply explaining the source code of a problem solution) often lead to unsatisfactory learning outcomes because the question of how a program was created remains unanswered. An improved approach considers "blueprints" for every technique taught to students. By allowing room for practice, letting students analyze required input and output data, the structure of a solution, and using beginner-friendly programming languages, better course performance was achieved. With regard to programming skills among students, a high diversity of skill levels can be observed. As an example (individual qualification of the student notwithstanding), it is assumed that a student with a bachelor's degree in economics working in accounting will need to rely on instructor directions to a greater extent compared to a student with a bachelor's degree in computer science working as a software architect. At the same time, programming courses within a course of study do not differentiate between these different types of students. In order to adequately assess the level of developer ability, i.e. the assumed level of support required (or desired), no 'official' framework comparable to, e.g., the Common European Framework of Reference for Languages (CEFR), could be identified in the course of the research. Nevertheless, there are various framework concepts in which peer reviews have not been carried out but which are intended to remedy this lack of a common framework. As an example, Poss [22] adapts the CEFR to programming skills (ranging from level A1 for a "basic" to level C2 for a "proficient" user) and allows assessment of competencies such as writing, explaining or reusing code or troubleshooting, bundled in the categories "writing", "understanding" and "interacting".

III. Related Work

As we base our approach on the results of Phung et al. [4], this work comes closest to our approach. Within their work, a study was conducted to compare SOTA generative AI and LLMs for a comprehensive set of Python programming scenarios. To this end, two models, namely ChatGPT (based on GPT-3.5) and GPT-4, were systematically evaluated, and their performance was compared with human tutors. Five introductory Python programming problems and real buggy programs from an online platform were used to evaluate performance [4]. The results showed that GPT-4 drastically outperforms ChatGPT (based on GPT-3.5) and comes close to the performance of human tutors in several scenarios but struggles, especially in the scenarios of grading feedback and task synthesis [4].

Another related work is by Zamfirescu-Pereira et al. [23], where an interactive LLM-based help assistant integrated in Visual Studio (VS) Code for programming was presented. As part of the work, the possibility of using GPT-4 as a pedagogically useful homework aid for students of an introductory course in computer science was investigated, also taking into account the programming language Python [23]. The GPT-4-based assistant adapts the code developed by the students into a prompt, thereby supporting the pedagogical goals and avoiding direct solutions. As a result, it was found that the LLM-based assistant can recognize the conceptual difficulties of the students and offers suggestions and code templates in a pedagogically appropriate way [23]. On the other hand, correct student code was sometimes inappropriately labeled as incorrect, and students were sometimes urged to use correct but inappropriate approaches for the lesson, which can lead to long and frustrating solutions [23].

Another paper dealing with education through LLMs is by Su and Yang [5], proposing a potential framework for the use of GenAI in education. The authors emphasize the versatility of LLMs in education, as they can be used to create virtual tutors, answer student questions, and provide personalized learning experiences. Their approach, the IDEE framework ("identifying desired outcomes", "determining the appropriate level of automation", "ensuring ethical considerations", "evaluating effectiveness"), can be used as a guide for evaluating the effectiveness of GenAI in education [5].

A. *Delimitation of own work*

As shown, there are similar works in the field of research, but they pursue different intentions, which is why we distinguish ourselves from them as follows. Fundamentally, we differ from Phung et al. [4] and Zamfirescu-Pereira et al. [23] in the programming language considered, as we base our research on the R programming language and not on Python. Phung et al. [4] themselves point out the limitation that they restrict their work to Python programming and that it would be interesting to conduct a similar study for other programming languages. We explicitly address this research gap in our work. Furthermore, we adapt the original methodology of Phung et al. [4] to the R programming language, which further expands the possibilities horizon with respect to LLMs and ITS. It should also be noted that, in addition to GPT-3.5 and GPT-4, we also include PaLM 2 via Google Bard to include another comparative model. The comparison of different SOTA LLMs also distinguishes us from the work of Zamfirescu-Pereira et al. [23], as they only focused on GPT-4. Furthermore, our aim is to investigate the suitability of LLMs for the individual teaching of R programming, while Zamfirescu-Pereira et al. [23] investigate support potentials for individual homework. When considering the work of Su and Yang [5], our goal differs as we do not aim to create a theoretical framework, but to analyze the performance within the application of SOTA ITS approaches with the integration of LLMs to improve programming education. Consequently, the objective differs since our work is at the application level, while Su and Yang [5] focus on a theoretical level.

IV. Methodology: Leveraging Generative AI for R tutoring

The literature found in the previous chapter demonstrates the potential of contemporary GenAI-based ITS to improve student learning experiences. For the purposes of our study, the study conducted by Phung et al. [4] is especially of interest. It addressed limitations found in earlier work, e.g., where AI models no longer available to the public were evaluated, and compared the performance of two LLMs (GPT-3.5 and GPT-4), against human tutors. The evaluation was based on five basic programming exercises in Python, such as string manipulation (palindrome check, string merging) or arithmetic tasks (Fibonacci sequences, greatest common divisor), and was conducted over six domains (for example, the repair of "buggy" code or creating novel programming tasks). Since satisfactory performance of the LLMs, especially GPT-

4, was observed, the results lend themselves to exploring the applicability of GenAI-based LLMs for R.

We thus seek to conduct an experimental study where the performance of three current SOTA LLMs (GPT-3.5 (via ChatGPT), PaLM 2 (via Google Bard) and GPT-4 (in the version tailored to Bing AI, as usage via ChatGPT requires a monthly fee)) is evaluated over five R programming exercises within four domains. Subsequently, possibilities for evaluating ITS in an institutional framework, i.e., in relation to students and teaching professionals, are outlined.

The three LLMs selected are accessed through the free of charge user interfaces to imitate the behaviour of actual students who are studying at university. The assumption is that students would choose to utilize AI systems, which are free of charge and available either directly or through a simple registration form.

By following this methodology, our study seeks to contribute to the ongoing discourse in the field, exploring the applicability of GenAI-based ITS in other programming languages. The following five programming problems P1 ... P5 were considered. The corresponding original prompts are as follows:

P1: **Greatest Common Divisor**:
`Given two positive integers A and B, find GCD of A and B.`

P2: **Fibonacci**:
`Given a positive integer N. Calculate the Fibonacci series till the number N. If N is a part of the series, include N as well.`

P3: **Divisors divisible by 3**:
`Given an integer N, find the number of divisors of N that are divisible by 3.`

P4: **Palindrome check**:
`Given a string S, check if it is palindrome or not.`

P5: **Merge Strings**:
`Given two strings S1 and S2 as input, the task is to merge them alternatively i.e. the first character of S1 then the first character of S2 and so on till the strings end.`

The exercises presented above were evaluated within the following four scenarios:

1) **Program repair**: the ITS automatically corrects faulty code.
2) **Hint generation**: the ITS generates a hint directed at the user, highlighting the nature of a certain fault within the code (without providing a direct solution).
3) **Contextualized explanation**: the ITS returns an explanation of a specific part of a program.
4) **Task generation**: the ITS creates a number of programming exercises based on criteria specified by the user.

An overview of the evaluation methodology is given in Figure 1.

A. *Methodology deviations and adaption*

While a high similarity between R and Python has been described in the literature, various adjustments to the original methodology used by Phung et al. [4] were required.

First, the scenarios 'pair programming' and 'grading feedback' are not included in our study, as proper evaluation of these scenarios with respect to the respective performance indicators developed by the authors was not possible due to a lack of sufficiently complex exercises within the repository containing training exercises (see below).

The study by Phung et al. [4] is based mainly on data taken from the "GeeksForGeeks" (GFG) platform. This platform offers coding challenges for various programming languages, e. g. Python. The users of the platform can solve tasks within a given code template, which is then automatically evaluated, and the results are public with their corresponding state (correct or false). Regarding our study, the methodology used by Phung et al. [4] cannot be fully replicated, as only coding tutorials for R are available on the platform, rather than (automatically verifiable) programming exercises. To verify the LLM output, RStudio was used, i. e. by running the prompt output manually.

The lack of interactive R exercises also applies to the code template used to solve the programming problems given. Therefore, the GFG Python code template was adapted to the R language. As interactive training of the R language is not possible on GFG, and no other source of "intentionally faulty" R code could be found, random faulty Python submissions were taken from the platform and translated into R code since the corresponding fault, i. e. inconsistent variable naming, was compatible. Using this approach, five faulty example source codes could be created for each problem.

V. Experimental Study: Evaluating a GenAI-based ITS for R Programming Education

A. *General problem-solving abilities of the LLMs*

Before prompting LLMs with queries based on the scenarios outlined before, two queries were used for each problem to evaluate and confirm the general problem-solving capability of the LLMs. The first prompt contained the problem as seen on GFG, together with the adapted code template to be filled. The second prompt only contained the task description, leaving the most efficient path to the solution to the LLM:

Of the compared LLMs, GPT-3.5 and GPT-4 were able to solve all five tasks in both cases. For the "Fibonacci" task, a regeneration of the answer was required to obtain the correct solution when using GPT-3.5, as the first answer contained a syntax error (wrong character) within the provided source code. For the prompt containing only the problem description, the provided solutions were, in some cases, shorter than the given template.

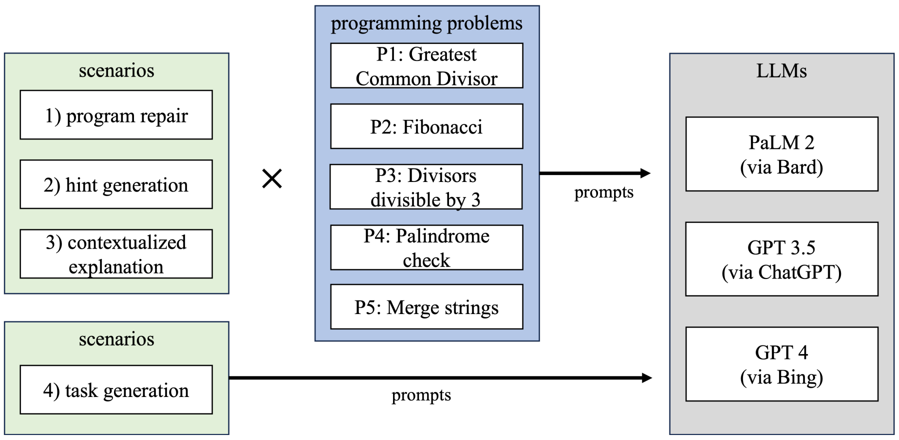

Fig. 1. Evaluation methodology: In order to evaluate the scenarios *program repair*, *hint generation*, and *contextualized explanation*, five pre-defined programming problems were used to create the queries. For the scenario *task generation*, the LLMs were prompted for tasks of three difficulty levels and two specified problem classes.

PaLM 2 (Google Bard) was not able to solve the tasks. When given the task without the code template, Python code was returned, although the prompt asked for R source code.

B. *Scenario 1: Program repair*

The first problem to be evaluated is the repair of buggy example programs for each problem. The LLMs are provided with a prompt containing the problem description, the faulty source code, and the request to fix the code with minimal changes according to the problem description. In each of the five problems, five different bugs were included and given to the LLMs, resulting in 25 queries made to each LLM. In the event that the LLMs' first response (i.e. the specified code) did not run as expected, the LLM was asked to revise it again, which was done, for example, with the input "The code still produces incorrect results".

In order to allow comparability, the prompts from [4] were used and adapted. The following prompt template was used, where `programming problem` and `code here` were replaced with the P1 ... P5 prompt texts as well as erroneous source code samples.

Listing 1. Prompt for scenario 1 (Program repair)

```
I'm working on an R programming problem. The
    current program below is not working well.

Can you help in fixing this program with as
    few changes as possible? Below I first
    provide the problem description and then
    the current buggy program.

(programming problem)

Buggy Program:

(code here)

Can you fix the above buggy program? Modify
    only the code within the function. Make
    sure that you make minimal possible
    changes needed to fix the program.
```

Evaluation. Points are awarded to LLMs successfully fixing a program: two points are given if the code was corrected and leads to the correct results according to the problem, and an additional point is given if no second prompt was required to achieve the correct solution.

Result. The table below provides the detailed results of the study per problem and LLM used. The highest possible performance score is 75, i.e. 15 points per problem over five problems. GPT-3.5 achieves 69 points, with very good performance over all problems. While GPT-4 exhibited perfect results in P1 to P3, the performance dropped with P4 and P5, as wrong results were observed or additional packages not relevant considering the simplicity of the problem were required. PaLM 2 provided mostly faulty code and, therefore, received a low score.

TABLE I
LLM SCORES WITHIN THE PROGRAM REPAIR SCENARIO

Problem	GPT-3.5	GPT-4	PaLM 2
P1 (of 15)	15	15	10
P2 (of 15)	13	15	0
P3 (of 15)	13	15	0
P4 (of 15)	14	9	0
P5 (of 15)	14	6	8
Total (of 75)	69	60	18

C. *Scenario 2: Hint generation*

To examine LLM performance for hint generation, the LLMs are prompted for an explanation about the bugs present within the provided source code and, as a second step, asked for a precise hint to fix one of the bugs. The problem description is also provided. For each problem/buggy program, one prompt was passed to the three different LLMs, resulting in 25 prompts for each LLM. In this scenario, the LLMs are only allowed one attempt at solving the problem. The following prompt was passed to the LLMs: w

Listing 2. Prompt for scenario 2 (Hint generation)

```
I'm working on an R programming problem. The
    current program below is not working well.

Can you help by giving a hint? Below I first
    provide the problem description and then
    the current buggy program.

(programming problem)

Buggy Program:

(code here)

(1) Can you describe the bug(s) in this
    program and the required fixes?
(2) Can you provide a concise single-sentence
    hint about one bug in this program?

The hint should not be too detailed as I want
    to think about the fixes by myself.
    However, the hint should not be too
    abstract, as I need some help.
```

Evaluation. We evaluate the hint generation scenario based on three criteria: (1) Bugs were pointed out correctly (the number of bugs pointed out was not considered), (2) an understandable hint was provided, and (3) the hint was not too detailed (e. g. no full source code was provided). Each of the criteria is rated with one point, leading to a maximum of 3 points per problem. For the second criterion, the authors rated 'understandability' as no automated and unbiased means of evaluation were available.

Result. The table below provides the detailed results of the study per problem and LLM used. The highest possible performance score is 75, i.e. 15 points per problem over five problems. GPT-3.5 performed best and was awarded 62 points. It was observed that all LLMs made mistakes in some cases, such as providing the full source code, providing incorrect hints or giving a hint which was longer than one sentence.

TABLE II
LLM SCORES WITHIN THE HINT GENERATION SCENARIO

Problem	GPT-3.5	GPT-4	PaLM 2
P1 (of 15)	15	14	5
P2 (of 15)	15	10	6
P3 (of 15)	12	8	5
P4 (of 15)	9	10	5
P5 (of 15)	11	10	5
Total (of 75)	62	52	26

D. Scenario 3: Contextualized explanation

The third scenario evaluates the contextualized explanation of parts of a program. The LLMs are prompted to provide an explanation of a single line of source code (LOC) within a full program. The problem, the (working) source code, and the specific LOC are given. For each problem, each LLM was given one prompt (without any regenerations or additional prompts, should the explanation fail).

Listing 3. Prompt for scenario 3 (Contextualized explanation)

```
I'm trying to understand a given program for
    an R programming problem.

Can you help by explaining a specific part of
    this program? Below I first provide the
    problem description, then the program, and
     then a specific part of this program.

(programming problem)

Program:

(code here)

Specific part:

(code here)

Can you provide a detailed explanation about
    the specific part above in the context of
    the whole program?
```

Evaluation. For this scenario, we compare the results of each of the three LLMs with each other and verify if the explanation is correct and sufficiently informative. A maximum of four points is awarded. The "best" explanation, i.e. the one closest to the goal of the scenario ("specific part in the context of the whole program") is awarded 3 points, the second best 2 points and the least good is awarded 1 point. The second parameter is checked based on the conciseness of the explanation – 1 point is awarded for correct and complete explanations, and zero points if the explanation is too unspecific.

Result. The table below provides the detailed results of the study per problem and LLM used. The highest possible performance score is 20, i. e. 4 points per problem over five problems. GPT-4 achieves full marks, followed by GPT-3.5 and PaLM 2. In general, GPT-4 completely followed the prompt and provided an explanation of the role of the LOC within the program. GPT-3.5 presented a detailed explanation of the whole program, but the given LOC was merely mentioned within the program. PaLM2 only provided general explanations, often limited to reflecting the problem description, e.g. "In the context of the whole program, this line is used to find the number of divisors of N that are divisible by 3", which defies the goal of this scenario.

E. Scenario 4: Task creation

The fourth scenario we evaluated is task generation. The LLMs were prompted to suggest five programming tasks (problems) to be solved by a student; additionally, a target difficulty (easy/medium/hard) was given. Furthermore, two tasks considering recursion and string manipulation were prompted. In both cases, the solution must not be provided. Note that the

TABLE III
LLM SCORES WITHIN THE CONTEXTUALIZED EXPLANATION SCENARIO

Problem	GPT-3.5	GPT-4	PaLM 2
P1 (of 4)	3	4	1
P2 (of 4)	3	4	1
P3 (of 4)	3	4	1
P4 (of 4)	3	4	1
P5 (of 4)	3	4	1
Total (of 20)	15	20	5

design of this scenario is not based on the study by Phung et al. [4].

Listing 4. Prompt for scenario 4 (Task creation)

```
4.1: Exercises with a specified skill level
I'm a student learning the R programming
    language. I would like to get some
    programming excercises.

Please create 5 programming tasks without
    providing the solution. The difficulty
    level for the tasks should be (desired
    skill level, i. e. easy, medium, hard).

4.2: Exercises on a specified topic
I'm a student learning the R programming
    language. I would like to get some
    programming excercises.

I'm not good with (topic, i. e. recursion,
    string manipulations). Please generate a
    task that I can use to practice. Don't
    provide the solution.
```

Evaluation. For the first part of the scenario, we check each of the five generated programming tasks for their difficulty (easy, medium, hard). For each task corresponding to that level, we award 1 point, leading to a maximum of 5 points for each of the three prompts. We evaluate the tasks based on the difficulty according to GFG and their complexity (e.g., requiring multiple functions and/or additional packages). If a solution is provided, 0 points are awarded. Regarding specific tasks, 2 points are awarded for a task appropriate for the student's weakness.

Result. The table below provides the detailed results of the study per problem and LLM used. The highest possible performance score is 19. In both parts of the scenario, GPT-3.5 performed best. While all three LLMs offered a correct selection of easy and specific tasks (some of them even overlapped with problems P1 to P5), the list of medium and hard tasks was also found to contain easy tasks. PaLM 2 delivered solutions for medium and hard problems as well as for specific tasks, which is against the goal of this scenario. In particular, GPT-4 focused the tasks more closely on unique R data structures, i.e., "Write a function that takes a data frame and a column name and returns the frequency table of that column.".

TABLE IV
LLM SCORES WITHIN THE TASK CREATION SCENARIO

Problem	GPT-3.5	GPT-4	PaLM 2
easy (of 5)	5	5	5
medium (of 5)	3	3	0
hard (of 5)	4	0	1
recursion (of 2)	2	2	0
string manipulation (of 2)	2	2	0
Total (of 19)	16	12	6

F. Score summary

The following table gives an overview of the total score achieved by the three LLMs:

TABLE V
TOTAL LLM SCORES OVER ALL SCENARIOS

Problem	GPT-3.5	GPT-4	PaLM 2
P. repair (of 75)	69	60	18
Hint gen. (of 75)	62	52	26
C. explan. (of 20)	15	20	5
T. creation (of 19)	16	12	6
Total (of 189)	162	144	55

As the table shows, GPT-3.5 is seen to exhibit the best performance, followed by GPT-4 and trailed by PaLM 2 by a wide margin. As such, we would like to affirm our initial research hypothesis that Generative AI is indeed able to serve as a tool for R programming instruction, albeit on a purely technical level. In the next chapter, we discuss our results with regard to our initial findings concerning student skills and evaluation within an institutional setting.

VI. DISCUSSION

The evaluation has shown that current AI language models generally exhibit satisfactory performance regarding R programming tasks, albeit with varying performance (i. e. the observed subpar performance of PaLM 2). However, these results do not indicate whether such a system can adequately be used within an institutional setting, i. e. by student learners. As outlined in Section 2, student skill levels will need to be considered in addition to implementing acknowledged architecture patterns and evaluating student acceptance.

Nonetheless, relevant evaluation criteria were identified in the literature. Sykes [24] designed an ITS (for learning Java) based on the ACT-R framework (Adaptive Control of Thought-Rational). Here, four principles need to be considered: (1) defining the target cognitive model, i. e. the educational goal, (2) communication between the ITS and the student (requiring students to construct the knowledge for themselves via experimental learning), (3) maximizing opportunities for practice, as this is a significant factor in student success, and (4) error remediation techniques avoiding "error states", when a student spends too much time on error correction. Furthermore, Sykes [24] used qualitative evaluation in the form of interview sheets, considering the following criteria: usefulness, benefit, improvement over a traditional classroom setting, ease of tutoring style, enjoyability and an overall improved learning experience. Abu-Naser et al. [19] used similar criteria in the evaluation of their Java-based ITS, i. e. quality of the ITS design, the recommendation of use, or the desire for adaptation to other courses. An important facet beyond the performance of the ITS itself is student acceptance, as outlined by Zhai et al. [15]. Huang et al. evaluated an ITS for math problems based on a Technology Acceptance Model (TAM) with respect to usefulness and ease of use [25]. The results show that students perceive usefulness as the more important factor, especially when the knowledge is relayed by an agent, i. e. an avatar. However, according to Zhai et al. [15], there is a risk of negative attitude towards AI techniques if they are not used appropriately.

Due to the nature of the tools used to interface with the LLMs evaluated, i.e. ChatGPT, Google Bard or Bing AI, no dedicated tool observing the architecture guidelines outlined in the previous chapter had to be considered. On the other hand, as, according to Zografos and Moussiades [26], GPT is an open domain system that can enter dialogues on various topics, it must be limited to specific learning tasks at a time. This can be achieved through the use of the ChatScript platform, which acts as a 'guiding module'. As such, a "real-world" deployment of our ITS would need to be encapsulated within an application conforming to one of the widely recognized architecture patterns previously outlined and limited in its functionality therein that it cannot be used for general queries, such as ChatGPT or Bing AI. Moral or ethical implications of our development, as outlined by, for example, Matias and Zipitria [12] in the form of algorithmic bias, influence of AI on instructors, and criteria such as fairness or transparency, will need to be considered, yet are not within the scope of this study and lend themselves to consideration in future work, for example by referencing the FATE model (Fairness, Accountability, Transparency and Ethics), a sociotechnical model for ethical issues in AI [12]. Furthermore, these issues are assumed to concern all the domains in which contemporary AI models are used beyond education.

VII. Limitations and Future Work

Our methodology is based on the work of Phung et al. [4], who conducted an evaluation based on human tutors, which was not considered in our work due to resource constraints. Instead, we performed tasks that required human evaluation based on the authors' knowledge (e.g., by manually implementing instructions to correct a buggy program in RStudio). However, it is conceivable that further assessments will be carried out by human tutors in the future, which would allow for expert assessment and potentially additional perspectives. In addition, as mentioned in the discussion section, a comparative analysis of our ITS and human tutors and the acceptance of the use of our ITS among students have not yet been conducted, which could also be considered for future work and provide further insight into the potential implementation of our proposed ITS.

The rapid development of LLMs promises further improvement of result quality in the (near) future. The poor result quality observed in PaLM 2, for example, is based on a version which was publicly available at the time of writing. Over the course of research, we conducted a second evaluation with samples from our prompt catalogue several months later and observed an improved quality of results, although the results stated here are based on our initial iteration. Future research should thus encompass ways to incorporate improved LLM versions in ITS.

Although AI is increasingly entering our education, Rachha and Seyam [18] report on several disadvantages of using AI in education with regard to the emerging paradigm of explainable AI (XAI) [27], [28]. Due to the "black box" nature of complex algorithms such as LLMs, users face pitfalls such as lack of transparency, potentially biased results of the system, and difficulties adapting to new circumstances. XAI is therefore seen as a prerequisite, as it builds trust and may be required in certain jurisdictions, e.g., to comply with the European General Data Protection Regulation (GDPR). Although the XAI-ED framework developed by Khosravi et al. [29] is described as a possible solution to the above pitfalls, the need for a unified framework is highlighted.

Another possibility for further research is the investigation of other programming languages, which was also discussed by Phung et al. [4] and forms the basis of our work. We have shown that the methodology is adaptable to other programming languages and that the comparison of the functionality of SOTA LLMs can be implemented, which is why it is conceivable to investigate other popular programming languages such as C++ or Kotlin. One of the main tasks here will be the definition of the output data, as we already had to make corresponding adjustments to the R queries, which limits the comparability across programming languages and can be particularly important for younger or less popular ones.

We would also like to note that, as the fourth evaluation scenario has shown, our ITS is able to adapt to individual skill levels, i.e. problems with different levels of difficulty can be generated. On the other hand, the assessment of programming skills is not based on a standardized framework. At the moment, students have to self-assess and prompt our ITS accordingly. In the future, it would be conceivable to use an integrated self-assessment to enable the ITS to offer the students suitable exercises automatically.

VIII. Conclusion

The need for powerful Intelligent Tutoring Systems arises in light of unsatisfactory interaction quality between students learning a new programming language and instructors needing to deal with limited interaction time and diverse programming skill levels among students entering programming courses. Contemporary literature shows that ITSes powered by modern AI techniques – here, Large Language Models, are able to adequately provide instructions to (student) users learning a programming language such as Python, yet the need for investigating LLM performance regarding additional, less popular programming languages was pointed out. In this work, we investigated whether publicly available LLMs are sufficiently powerful to provide instructions for the R programming language and thus can take on the role of a (human) tutor. For this purpose, we have given prompts concerning four tutoring scenarios and five programming problems to three LLMs: GPT-3.5 via ChatGPT, PaLM 2 via Google Bard, and GPT-4 via Bing. We observed that GPT-3.5 showed the best performance, although GPT-4, for example, was able to achieve full marks in the contextualized explanation domain. On the basis of our results, we recommend several directions for future work, with the application of our findings within an institutional setting (i. e. benchmarking our results against human tutors or developing a student self-assessment module for providing skill-appropriate tasks) being the most prominent one.

References

[1] A. Bey and R. Champagnat, "Toward a Smart Tool for Supporting Programming Lab Work," in *Augmented Intelligence and Intelligent Tutoring Systems*, 2023, vol. 13891, pp. 290–297.

[2] H. Crompton and D. Burke, "Artificial intelligence in higher education: The state of the field," *International Journal of Educational Technology in Higher Education*, vol. 20, no. 1–22, p. 22, 2023.

[3] M. Á. Escotet, "The optimistic future of Artificial Intelligence in higher education," *PROSPECTS*, pp. 1–10, 2023.

[4] T. Phung, V.-A. Pădurean, J. Cambronero, S. Gulwani, T. Kohn, R. Majumdar, A. Singla, and G. Soares, "Generative AI for Programming Education: Benchmarking ChatGPT, GPT-4, and Human Tutors," in *Proceedings of the 2023 ACM Conference on International Computing Education Research*, vol. 2, 2023, pp. 41–42.

[5] J. Su and W. Yang, "Unlocking the Power of ChatGPT: A Framework for Applying Generative AI in Education," *ECNU Review of Education*, vol. 6, no. 3, pp. 355–366, 2023.

[6] Kuhn and Max, "Building predictive models in r using the caret package," *Journal of Statistical Software*, vol. 28, no. 5, p. 1–26, 2008. [Online]. Available: https://www.jstatsoft.org/index.php/jss/article/view/v028i05

[7] A. Theissler and P. Ritzer, "EduML: An explorative approach for students and lecturers in machine learning courses," in *2022 IEEE Global Engineering Education Conference (EDUCON)*, 2022, pp. 921–928.

[8] H. Wickham, M. Çetinkaya-Rundel, and G. Grolemund, *R for data science*. " O'Reilly Media, Inc.", 2023.

[9] T. Mailund, *R Data Science Quick Reference: A Pocket Guide to APIs, Libraries, and Packages*. Apress, 2019.

[10] IEEE Spectrum, "Top Programming Languages 2022," https://spectrum.ieee.org/top-programming-languages-2022, 2021, accessed: 2023/08/26.

[11] IBM Cloud Team, "Python vs. R: What's the Difference?" https://www.ibm.com/blog/python-vs-r/#, 2021, accessed: 2023/08/26.

[12] A. Matias and I. Zipitria, "Promoting Ethical Uses in Artificial Intelligence Applied to Education," in *Augmented Intelligence and Intelligent Tutoring Systems*. Springer Nature Switzerland, 2023, vol. 13891, pp. 604–615.

[13] Y. Jiang, X. Li, H. Luo, S. Yin, and O. Kaynak, "Quo vadis artificial intelligence?" *Discover Artificial Intelligence*, vol. 2, no. 1, pp. 1–19, 2022.

[14] H. Hirsch-Kreinsen, "Artificial intelligence: A "promising technology"," *AI & SOCIETY*, pp. 1–12, 2023.

[15] X. Zhai, X. Chu, C. S. Chai, M. S. Y. Jong, A. Istenic, M. Spector, J.-B. Liu, J. Yuan, and Y. Li, "A Review of Artificial Intelligence (AI) in Education from 2010 to 2020," *Complexity*, vol. 2021, pp. 1–18, 2021.

[16] M. Yazdani, "Intelligent tutoring systems: An overview," *Expert Systems*, vol. 3, no. 3, pp. 154–163, 1986.

[17] C. Troussas, C. Papakostas, A. Krouska, P. Mylonas, and C. Sgouropoulou, "Personalized Feedback Enhanced by Natural Language Processing in Intelligent Tutoring Systems," in *Augmented Intelligence and Intelligent Tutoring Systems*, 2023, vol. 13891, pp. 667–677.

[18] A. Rachha and M. Seyam, "Explainable AI In Education : Current Trends, Challenges, And Opportunities," in *SoutheastCon*, 2023, pp. 232–239.

[19] S. Abu-Naser, A. Ahmed, N. Al-Masri, A. Deeb, E. Moshtaha, and M. Abu Lamdy, "An Intelligent Tutoring System for Learning Java Objects," *International Journal of Artificial Intelligence & Applications*, vol. 2, no. 2, pp. 68–77, 2011.

[20] P. Brusilovsky, A. Kouchnirenko, P. Miller, and I. Tomek, "Teaching programming to novices: A review of approaches and tools," in *Proceedings of ED-MEDIA 94–World Conference on Educational Multimedia and Hypermedia*, 1994, pp. 103–110.

[21] M. Crestani and M. Sperber, *Erfolgreich Programmieren Lernen Und Lehren*. Franzbecker, 2012.

[22] R. Poss, "Programming skills self-assessment matrix," https://dr-knz.net/programming-levels/prog-skill-matrix.pdf, accessed: 2023/08/26.

[23] J. D. Zamfirescu-Pereira, L. Qi, B. Hartmann, J. DeNero, and N. Norouzi, "Conversational Programming with LLM-Powered Interactive Support in an Introductory Computer Science Course," *NeurIPS'23 Workshop on Generative AI for Education (GAIED)*, pp. 1–10, 2023.

[24] Edward Sykes, "Qualitative Evaluation of the Java Intelligent Tutoring System," *Systemics, Cybernetics and Informatics*, vol. 3, no. 5, pp. 1–12, 2006.

[25] H. Huang, Y. Chen, and P.-L. P. Rau, "Exploring acceptance of intelligent tutoring system with pedagogical agent among high school students," *Universal Access in the Information Society*, vol. 21, no. 2, pp. 381–392, 2022.

[26] G. Zografos and L. Moussiades, "A GPT-Based Vocabulary Tutor," in *Augmented Intelligence and Intelligent Tutoring Systems*, C. Frasson, P. Mylonas, and C. Troussas, Eds. Cham: Springer Nature Switzerland, 2023, vol. 13891, pp. 270–280.

[27] R. Guidotti, A. Monreale, S. Ruggieri, F. Turini, F. Giannotti, and D. Pedreschi, "A survey of methods for explaining black box models," *ACM computing surveys (CSUR)*, vol. 51, no. 5, pp. 1–42, 2018.

[28] A. Theissler, F. Spinnato, U. Schlegel, and R. Guidotti, "Explainable AI for time series classification: a review, taxonomy and research directions," *IEEE Access*, pp. 1–25, 2022.

[29] H. Khosravi, S. B. Shum, G. Chen, C. Conati, Y.-S. Tsai, J. Kay, S. Knight, R. Martinez-Maldonado, S. Sadiq, and D. Gašević, "Explainable Artificial Intelligence in education," *Computers and Education: Artificial Intelligence*, vol. 3, pp. 1–22, 2022.