Journal of Computer Assisted Learning

WILEY

Journal of Computer Assisted Learning

ORIGINAL ARTICLE OPEN ACCESS

You're (Not) My Type- Can LLMs Generate Feedback of Specific Types for Introductory Programming Tasks?

Dominic Lohr[1] | Hieke Keuning[2] | Natalie Kiesler[3]

[1]Friedrich-Alexander-Universität Erlangen, Nürnberg, Germany | [2]Utrecht University, Utrecht, The Netherlands | [3]Nuremberg Tech, Nürnberg, Germany

Correspondence: Dominic Lohr (dominic.lohr@fau.de)

Received: 27 March 2024 | **Revised:** 23 July 2024 | **Accepted:** 2 December 2024

Funding: The authors received no specific funding for this work.

Keywords: ChatGPT | feedback | feedback classification | feedback types | GenAI | generative AI | GPT-4 | introductory programming | large language models

ABSTRACT

Background: Feedback as one of the most influential factors for learning has been subject to a great body of research. It plays a key role in the development of educational technology systems and is traditionally rooted in deterministic feedback defined by experts and their experience. However, with the rise of generative AI and especially large language models (LLMs), we expect feedback as part of learning systems to transform, especially for the context of programming. In the past, it was challenging to automate feedback for learners of programming. LLMs may create new possibilities to provide richer, and more individual feedback than ever before.

Objectives: This article aims to generate specific types of feedback for introductory programming tasks using LLMs. We revisit existing feedback taxonomies to capture the specifics of the generated feedback, such as randomness, uncertainty and degrees of variation.

Methods: We iteratively designed prompts for the generation of specific feedback types (as part of existing feedback taxonomies) in response to authentic student programs. We then evaluated the generated output and determined to what extent it reflected certain feedback types.

Results and Conclusion: This study provides a better understanding of different feedback dimensions and characteristics. The results have implications for future feedback research with regard to, for example, feedback effects and learners' informational needs. It further provides a basis for the development of new tools and learning systems for novice programmers including feedback generated by AI.

1 | Introduction

Learning to program involves much more than just understanding the syntax and semantics of a programming language. It is therefore not surprising that it has been characterised as a challenging process (Qian and Lehman 2017; Medeiros, Ramalho, and Falcão 2018; Luxton-Reilly et al. 2018; Kiesler 2024). Even experienced programmers regularly encounter errors in their code—a testament to the inherent complexity of programming. Due to experience and years of practice, programming experts have an arsenal of debugging techniques at their hands. Novice programmers, however, lack this experience, causing them to be more dependent on external support, for example, from educators, peers or tools. This is particularly true at the very beginning of the learning process.

Feedback—as a means of external support—is one of the most influencing factors of learning success (Hattie 2009). In the context of programming education, elaborated feedback has not yet been implemented at scale, for example, as a feature of

Summary

- What is already known about this topic?
  - Feedback is one of the most influential factors for learning.
  - Feedback in systems for learning to program is mostly simple feedback.
- What this paper adds?
  - A study of the capabilities of Large Language Models (LLMs) to generate specific types of (elaborated) feedback on programming submissions.
- Implications for practice and/or policy
  - Implementing LLMs in educational settings could enhance the feedback loop in programming courses.
  - Awareness among learners about LLMs' limitations is crucial, as their output may appear to look sensible, but can be misleading.
  - Educational tool developers should think about hybrid approaches that combine traditional test-driven methods with LLMs to offer more elaborated feedback and reduce the chances of misleading information.

learning environments (Jeuring et al. 2022). Common systems (e.g., Codewars, Codecademy, Python Tutor, W3Schools) often tend to provide simple feedback indicating the correctness of the code or pointing out errors without addressing the underlying causes or how to fix these errors (Jeuring et al. 2022). In contrast, adaptive feedback addressing the causes of errors and not just their manifestation in the code is considered more valuable by learners (Lohr and Berges 2021).

The recent emergence of advanced natural language processing techniques, generative AI (GenAI), and particularly the broad availability of large language models (LLMs), offers new opportunities for improving and elaborating feedback in programming education. Previous research has started to investigate the potential of LLMs to generate, for example, code explanations, enhanced error messages and thus formative programming feedback (Azaiz, Kiesler, and Strickroth 2024; Leinonen et al. 2023; Sarsa et al. 2022; MacNeil et al. 2023; Azaiz, Deckarm, and Strickroth 2023; Bengtsson and Kaliff 2023; Kiesler, Lohr, and Keuning 2023). However, these studies are usually based on simple help requests as phrased by students. Moreover, the rich diversity of feedback types with regard to its content has not yet been addressed.

In this article, we explore to what extent the feedback generated by LLMs can be controlled and how its quality can be assessed. Our experiments pursue the following goals: (1) design prompts to generate specific types of feedback from a specific feedback taxonomy for the context of programming (see (Keuning, Jeuring, and Heeren 2018)), and (2) evaluate the adequacy and quality of the generated feedback types. To address these goals, we used an iterative process to develop a prompt to generate different types of feedback. We used this prompt and a dataset of authentic student solutions (correct ones and with errors) as input, aiming to generate different types of feedback for these student solutions. The same dataset has been used in related work (Kiesler, Lohr, and Keuning 2023) to generate feedback, but this related study did not apply "prompt engineering techniques"—meaning the input was simple and the feedback generated by the LLM was characterised with regard to its content, quality and other criteria.

Our article is structured as follows. In Section 2, we describe related work on the role of feedback in learning in general, and how feedback has been automated in the context of computing education. We also outline recent research on the emergence of GenAI in computing education and summarize studies employing GenAI for the generation of feedback in programming education. We describe the method of our study in Section 3, which includes the selection of datasets with (erroneous) student programs, the prompt design process and the approach for characterizing the feedback output. We present and discuss the results in Section 4, and the limitations in Section 5. In Section 6, we explore the implications for educators, researchers, students and tool designers, which is followed by conclusions in Section 7.

2 | Background and Related Work

To contextualize the present study, we introduce the role of feedback as part of students' learning process before focusing on feedback types for programming tasks and the recent role of GenAI in programming education.

2.1 | The Role of Feedback in the Learning Process and the Programming Domain

Feedback has been the subject of extensive research in a great number of disciplines. According to a meta-analysis, it is one of the most influential factors for learning (Hattie 2009). At the same time, it is crucial to consider how feedback is delivered and presented (Narciss 2008). In general, the goal of giving feedback is to reduce the gap between the student's current performance and their desired goal (Hattie and Timperley 2007), and to modify their thinking and behavior (Shute 2008). Research shows the importance of giving detailed feedback, as opposed to simple messages only indicating whether a solution is correct or incorrect. Feedback may also be classified as *summative* (focusing on performance after submission to educators) or *formative* (focusing on the learning process during the execution of a task or a course). In this study, we refer to formative feedback provided to students trying to solve a task or assignment as part of their coursework.

According to Hattie and Timperley (Hattie and Timperley 2007), effective feedback should answer the questions 'Where am I going' (feed up), 'How am I going' (feed back) and 'Where to next' (feed forward). Each of these questions can be applied to four levels: (1) task, (2) process, (3) self-regulation and (4) self. Shute (Shute 2008) recommends formative feedback to be non-evaluative, supportive, timely and specific.

The literature distinguishes various dimensions for the design of feedback (Narciss 2008; Shute 2008). Narciss (Narciss 2008) outlines *function* (cognitive, meta-cognitive and motivational), *presentation* (timing, number of tries, adaptability, modality) and *content* of feedback. Each of these dimensions can influence the feedback's effects. Regarding content, Narciss classified feedback components for digital learning environments based on which

aspects of the instructional context the feedback addresses, such as task constraints, mistakes and procedural knowledge.

Narciss (Narciss 2006) summarizes feedback into *simple* and *elaborated* types. The latter provides more detailed information to the learner to analyze and improve their study. Keuning et al. (Keuning, Jeuring, and Heeren 2018) have applied and extended this categorization for the programming domain. Table 1 provides an overview of the different feedback types with a description of how they are applied to the context of programming. The feedback types we used to analyse the data are marked with an asterisk (*).

Keuning et al. (Keuning, Jeuring, and Heeren 2018) labeled the feedback of more than 100 available programming learning tools using these categories, showing that feedback provided by these tools (based on papers up until 2015) mostly focuses on pointing out mistakes. The authors omitted the simple feedback types in their study. However, the simple types of feedback have been used in many learning tools, for example, to indicate correctness or to show the fully correct solution. At the same time, other research has shown that binary feedback may not be beneficial to learning (Kyrilov and Noelle 2016).

The presented classification has been used in other studies (e.g., (Jeuring et al. 2022; Brocker and Schroeder 2023; Lohr et al. 2024; Kiesler 2023)). Jeuring et al. (Jeuring et al. 2022), for example, investigated the feedback types implemented in well-known online programming learning environments. They found that these environments mostly provide simple feedback and compiler or test-based feedback. Moreover, they found that educators as experts would provide different types of feedback, that is, more elaborate feedback types (see also (Lohr et al. 2024)).

The feedback types described by Keuning et al. (Keuning, Jeuring, and Heeren 2018) are closely related to the types of issues, errors, and misconceptions students may have or produce. Qian and Lehman's review of misconceptions in introductory programming (Qian and Lehman 2017) discusses issues that arise in three main categories related to programming knowledge (based on Bayman and Mayer (Bayman and Mayer 1988)), referring to syntactic, conceptual or strategic knowledge. Syntactic errors are often caught by the compiler. However, it is well known that students struggle with compiler error messages (Becker et al. 2019). Conceptual problems can be caused by misconceptions regarding mental models of code execution and computer systems (e.g., (Kiesler 2022a)). They are much harder to address but may be mitigated by certain feedback types (KC, KM subtypes on solution errors and test failures and KH error correction feedback). Strategic problems concern planning, writing and debugging programs as a solution to a novel problem. They require both syntactic and conceptual knowledge as a prerequisite.

TABLE 1 | Feedback Types (Keuning, Jeuring, and Heeren 2018).

Label	Name	Description
Simple types		
KR[a]	Knowledge of result	A simple indication of 'correct' or 'incorrect'. Keuning et al. define 'correct' in the context of a programming task as one or more of the following elements: (1) it passes all tests, (2) it is equal to a model program and/or (3) it satisfies one or more constraints.
KCR	Knowledge of correct result	A description or an indication of the correct response (e.g., providing the learner a model solution)
KP[a]	Knowledge of performance	Summative feedback on the achieved performance level regarding a set of tasks (e.g., '6 out of 10 tasks correct' or '75% correct'.)
Elaborate types		
KTC[a]	Knowledge about task constraints	Focuses on the task itself. This feedback could be about certain constraints (not using library functions, or enforcing a specific approach, such as recursion), or more general hints on how to approach the task without taking the student's current program into account.
KC[a]	Knowledge about concepts	Contains explanations on topics and concepts relevant to the exercise, or examples illustrating these concepts.
KM[a]	Knowledge about mistakes	Points out mistakes, by means of showing syntactic or semantic errors detected by a compiler (not exercise-specific), test cases that fail, or solution errors (logical or runtime). Other subcategories focus on style and performance-related issues. This content can be *basic*, containing a number (e.g., showing the number of failed tests), location or short identifier; or with a *detailed* description.
KH[a]	Knowledge on how to proceed	Hints on how to correct errors, next-step hints and hints on how to improve the style of a semantically correct program.
KMC	Knowledge about meta-cognition	Feedback related to problem-solving strategies and the student's awareness of their own performance.

[a]Used for data analysis.

In addition, non-functional aspects of programming have been gaining more attention in courses and assessments, such as code quality, style and performance. In other cases, a program might be error-free but incomplete, and the student needs help with the next step. Hence, students' informational needs can greatly vary.

2.2 | Using GenAI for Feedback in Programming Education

In the past decades, many different techniques have been applied to automatically generate feedback with the goal of supporting educators and learners at scale (e.g., in Massive Open Online Courses) (Keuning, Jeuring, and Heeren 2018; Messer et al. 2024; Paiva, Leal, and Figueira 2022). While automated testing is still one of the most common ways to generate feedback, several data-driven methods have emerged. However, with LLMs and GenAI tools becoming available to the public at the end of 2022, the possibility of generating a diverse range of feedback has become a reality and a true option for both educators and learners.

Accordingly, GenAI and LLMs have been extensively investigated within the last two years, especially in the context of programming education (Prather et al. 2023; Prather et al. 2024; Becker et al. 2024). Early articles focused on analyzing how well LLMs could solve well-known (introductory) programming problems (Finnie-Ansley et al. 2022; Denny, Kumar, and Giacaman 2023; Wermelinger 2023; Cipriano and Alves 2023; Kiesler and Schiffner 2023). Recently, researchers have shifted their focus towards investigating the use of LLMs to enhance teaching and learning how to program, and their integration into the classroom (Scholl, Schiffner, and Kiesler 2024; Scholl and Kiesler 2024; Prather et al. 2024). OpenAI's GPT-4 model's capabilities in identifying errors within programming code have been explored (Wu et al. 2023), as well as its performance in (beginner and intermediate) `Python` course exams (Savelka et al. 2023), answering various types of questions (Joshi et al. 2024), or for automatic program repair and refactoring (Ishizue et al. 2024).

Another potential application of LLMs is the generation of code explanations. MacNeil et al. (MacNeil et al. 2023) showed that students perceive generated line-by-line code explanations as helpful. In another study (Leinonen et al. 2023), students rated the code explanations of the LLM better on average than those created by students. Especially the level of understanding and accuracy was rated higher, and students did not show negative affection towards the generated, non-human feedback.

Due to GenAI tools and LLMs, it is thus possible to generate feedback for (novice) learners of programming. Several studies have investigated the feedback generated by LLMs. For example, Balse et al. (Balse et al. 2023) used final exam submissions to seven exercises as input to GPT-3 and evaluated the response on correctness and type of feedback. They found that 71% of the feedback was accurately checked as correct, and around 62% contained correct critiques and suggestions. The authors identified four themes in the feedback that mostly target the constructs on which feedback was given: 'generic feedback', 'variable initialization and updates', 'conditionals' and 'iteration'.

Pankiewicz and Baker (Pankiewicz and Baker 2023) implemented GPT-3.5 feedback inside an existing automated assessment tool, targeted at `C#` submissions to 38 exercises. The students rated the feedback on a Likert scale, and their performance, time-on-task and self-reported affective state were measured and compared with a control group that did not receive the hints. Almost half of the students were positive about the feedback, which was in Polish, but some students were less happy with the hints. The authors also found that the experimental group with GPT hints solved tasks faster and submitted correct solutions more often. When they had to solve problems without hints afterwards, they initially struggled, but caught up with the control group in the end. The prompt contained the Polish exercise description, the student's code, information about failed tests and compiler errors, and based on the latter, a prompt with instructions. An example of this prompt was given in the paper, which instructed not to give away the solution and only give hints, and to highlight certain elements (e.g., keywords, variable names, line numbers) using html-tags.

Related work by Kiesler et al. (Kiesler, Lohr, and Keuning 2023) generated feedback for various `Java` programming exercises using ChatGPT 3.5, and a very simple prompt (which students might use). The prompt consisted of the question 'What is wrong with my code?' followed by the student submission to an introductory programming task. The generated output was qualitatively analysed with regard to its content, quality, and other elements (see Table 2). The authors found that several LLM-generated feedback messages contained misleading information (Kiesler, Lohr, and Keuning 2023). Some of these were because the prompt did not contain any information about the task or special task constraints. For example, it was not permitted to use `Java` libraries, although this was occasionally requested or included in the LLM's feedback.

Roest et al. (Roest, Keuning, and Jeuring 2024) focused on generating next-step hints for introductory `Python` exercises. They iteratively developed a prompt for GPT 3.5 to generate a short hint on what to do next (i.e., when a student is working towards a solution). The prompt contained the task description, and specific keywords in a relatively short instruction. This prompt was then used in an online programming tool to deliver hints on the students' requests. A small experiment with students was conducted, as well as an evaluation of generated hints with experts, using the following 9 criteria: *type*, *level of detail*, *information*, *personalised*, *appropriate*, *specific*, *misleading information*, *tone* and *length*. Overall, the feedback from students and experts regarding the hints was positive. However, there were instances of misleading information, and the hints were found to lack detail, particularly as students neared the completion of the exercises. Xiao et al. (Xiao, Hou, and Stamper 2024) also explored hint generation. Specifically, they measured the effects of four levels of hints, from a generic text hint to concrete code solutions.

Azaiz et al. (Azaiz, Deckarm, and Strickroth 2023) characterised the feedback generated by GPT3 (text-davinci-003) for both correct and incorrect submissions to two Java programming exercises. They used a simple prompt along with the task description and the student code. They studied the responses' length (the difference between the two exercises was statistically

TABLE 2 | Characteristics of LLM feedback messages (Kiesler, Lohr, and Keuning 2023).

Label	Meaning	Value options
Content		
INFO	Requesting more information	Y/N
STYLE	Stylistic suggestion	Y/N
CAUSE	Textual explanation cause of error	Y/N
FIX	Textual explanation fix of error	Y/N
CODE	Code provided. A snippet could be a (few) statement(s), or an expression, but not a single variable name.	Y/Snippet/N
EXA	Illustrating examples	Y/N
Quality		
COMP	Code, if provided, compiles (or in case of a snippet, is correct)	Y/N/n.a.
MIS	Contains misleading information	Y/N
UNC	Expression of uncertainty	Y/N
PERS	It is personalised to the student's code, such as using their variable names or referring to their algorithm	Y/N
COMPL	It matches the task description	Y/N
Other		
META	Meta-cognitive elements	Y/N
MOT	Motivational elements	Y/N

significant), whether it correctly identified correct solutions (in 73% of the cases, comparable to Balse et al. (Balse et al. 2023) who employed the same model), and the characteristics of its content (code elements, personalization, style suggestions, correctness of suggestions, specification compliance and fault localization). Submissions were finally clustered into three categories: syntactically and functionally correct, syntactically correct but functionally not correct (SCFI, not meeting task requirements), and syntactically and functionally incorrect (SIFI). Several issues with the LLM-generated feedback are reported, such as giving suggestions that do not align with the requirements. The authors (Azaiz, Deckarm, and Strickroth 2023) conclude that 'currently, it is not good enough for summative grading'. However, they see the potential of LLMs to support teaching assistants in grading.

In a recent follow-up study by Azaiz et al. (Azaiz, Kiesler, and Strickroth 2024), GPT-4 Turbo's feedback in response to the dataset with the same 55 student programming submissions as in previous study (Azaiz, Deckarm, and Strickroth 2023) was evaluated. They qualitatively analysed the output with regard to its feedback content and structure, code representation, corrections and correction types, suggested optimizations and coding style, as well as inconsistencies and redundancies. They conclude that all of the feedback generated by GPT-4 is personalised and significantly better in terms of its accuracy compared with previous GPT versions (Azaiz, Kiesler, and Strickroth 2024). Moreover, they note that following all the LLM's suggestions and corrections would have resulted in fully correct solutions (except for two cases). At the same time, only 52% of the feedback and all of its details were fully correct and complete.

Nguyen and Allan (Nguyen and Allan 2024) present their experience with using GPT-4 to provide tiered, formative feedback to `Java` code and pseudocode in the context of a course on algorithms and data structures. Their goal was to elicit feedback on conceptual understanding, syntax and time complexity, as well as next-step hints or guiding questions. They analysed the feedback regarding its accuracy and usefulness and its correctness, concluding that GPT-4 generally provided accurate feedback, which was perceived as useful in guiding students' next steps. However, they also noted a varying performance across the output for the 113 submissions to four programming problems.

Phung et al. (Phung et al. 2024) have developed a new technique to provide feedback on buggy programs, incorporating symbolic data such as failed tests and a fixed program to generate LLM-based feedback, with a validation step to ensure higher quality. Koutcheme et al. (Koutcheme et al. 2024) have explored the capabilities of open-source LLMs to generate feedback and used GPT to automatically assess the feedback quality.

Finally, several studies have focused on using LLMs to enhance compiler error messages. Wang et al. (Wang, Mitchell, and Piech 2024) used GPT to enhance `Python` error messages. They conducted a large-scale randomised control trial in which they compared six different approaches, of which two approaches used GPT-3. They found that the GPT-enhanced messages were most effective; students resolved the error in fewer attempts, and they repeated the error less often. Taylor et al. (Taylor et al. 2024) present a tool that integrates an LLM into a `C` compiler, providing student-friendly error messages that help them solve the problem. They deployed the tool at a large scale and analysed the quality of the generated messages. Kimmel et al. (Kimmel et al. 2024) used GPT-4 to give feedback on compiler, runtime and logic errors, which they deploy in an automated assessment tool. They observed mixed opinions from students and argued that the UI design of such a tool influences how the feedback is perceived.

3 | Methods

3.1 | Research Motivation

Related research had the goal to generate 'general' feedback in response to a task, student solution or the question of how to proceed. These studies concluded that LLM-generated feedback

TABLE 3 | Exercise overview for prompt development and evaluation.

Name	Description	Concepts	Language	Used for
Physics	Write methods for the given thermal equations of state using the constants provided.	Class, methods, calculations, constants	Java	Prompt development, evaluation
Triangles	Write methods to classify a triangle (1) by its sides (equilateral, isosceles or scalene), depending on whether all three sides, exactly two sides, or no sides are equal, and (2) by its angles (acute, right or obtuse) depending on whether all three angles are less than 90°, one angle is exactly 90° or one angle is greater than 90°.	Class, methods, calculations, conditionals, enums	Java	Prompt development, evaluation
NegaFib	Compute the negative Fibonacci sequence for negative inputs without using classes or methods from the Java API.	Methods, conditionals, recursion	Java	Prompt development, evaluation
Fibonacci	Compute the n-th Fibonacci number using recursion.	Methods, conditionals, recursion	Java	Evaluation
Password	A function asking the user for a password until they enter the correct one (given as parameter).	Methods, parameters, loops, conditionals, input/output	Dart	Evaluation
MaxOf3	Print the largest of three numbers in the input.	I/O, conditionals	Python Kotlin, Java, C++	Evaluation

messages may contain misleading information, incorrect corrections and lack of details (Azaiz, Kiesler, and Strickroth 2024; Azaiz, Deckarm, and Strickroth 2023; Kiesler, Lohr, and Keuning 2023; Roest, Keuning, and Jeuring 2024) in response to student programming submissions. At the same time, LLM-generated feedback is personalised and can address several issues and aspects of an input (e.g., conceptual knowledge, the task itself, mistakes, hints on how to improve or even meta-cognitive strategies).

To date, learning environments for programming mostly provide a limited set of feedback types, and they hardly give elaborated, let alone personalised feedback (Jeuring et al. 2022). Therefore, it is the goal of this study to investigate how and to what extent we can generate specific, elaborated feedback types by using a state-of-the-art LLM (e.g., GPT-4).

In addition, currently available programming feedback classifications like the one by Keuning et al. (Keuning, Jeuring, and Heeren 2018) were developed at a time when feedback generation options were limited and complex to automate. In the LLM era, GenAI may provide us with a richer array of possibilities, implying the need to revisit this taxonomy to showcase these possibilities.

3.2 | Scope and Research Questions

This study focuses on giving feedback to students working on a single programming exercise while limiting the scope to the following aspects: (1) The student's program for which we want to provide feedback is (almost) complete. We do not consider partial solutions from the early phases of the problem-solving process. (2) The goal of this study is to critically analyze the input provided to the LLM and to identify suggestions on how to improve and expand the input in an educational setting. This scenario differs from related work using LLMs for code explanations, fault localization and repair for a professional setting, and enhancing existing compiler error messages. (3) We focus on formative, elaborated feedback, but also investigate some simple elements such as KR as a component of more extended feedback. The following research questions guide our study:

RQ 1. *To what extent can feedback of a specific type be generated using LLMs for a given (faulty) program?*

RQ 2. *What are the characteristics of the generated feedback?*

3.3 | (Re-)Use of Datasets

3.3.1 | Dataset Used for Prompt Development

To create a complete and comprehensive prompt, we, first of all, made sure that the prompt we intended to use would include all relevant information as input to the LLM. Therefore, we selected a dataset that encompassed all the information available about its setting and context. For example, the task description, a model solution, and templates (e.g., class skeleton) students may have received. Precisely, we used the dataset from a related

study, published at the 2023 Frontiers in Education conference (Kiesler, Lohr, and Keuning 2023). The respective data were gathered via weekly exercises in a (large) introductory programming course in `Java`. Three exercises were selected from this set (Physics, Triangles and NegaFib in Table 3). For all but the NegaFib exercise a template with a class skeleton containing empty methods was provided to the students. We made some changes to the Triangle task by merging its two subtasks into one task. This was because we wanted to prompt the LLM to generate the complete solution, but also all additional information (e.g., related programming concepts). This way, the use case was more similar to students solving the complete task as part of their coursework.

As an initial set for designing the prompt, we selected one student submission for each exercise that contained multiple errors, such as syntax errors, semantic errors and logic errors (for an overview, see Table 4).

3.3.2 | Dataset Used for Evaluation

To test the performance of the model with our prompt on a more diverse set of programming submissions, we expanded our dataset based on the following requirements for the exercises and corresponding solutions:

- Exercises should cover various types of tasks.
- Student solutions should be in different programming languages.
- Student solutions should have diverse types of errors.
- Student solutions should have different numbers of errors, ranging from zero to numerous.
- Student solutions should be reasonably complete.
- Some of the solutions should be correct to check if the generated feedback is also valuable for correct submissions.

We selected a total of 11 solutions to 6 different programming exercises from existing datasets (see Table 3) that had been used before in research (Lehtinen 2022; Kiesler 2022b; Kiesler 2022c). A new exercise, MaxOf3, was taken from the TaskTracker dataset (Lyulina et al. 2021). This dataset contains snapshots from 148 students solving 6 introductory programming tasks in 4 different programming languages. The data were collected within a plugin for IntelliJ-based IDEs. We selected the MaxOf3 exercise because there were both correct and incorrect submissions for it in various programming languages. The exercise was accompanied by a set of test cases (input/output) and a task description. As there was no model solution provided, we created one for each programming language.

3.4 | Prompt Design Process

Next, we outline the iterative process of evaluating the outcomes of each prompt and how we enhanced the instructions provided within each prompt. Five iterations were needed before arriving at a prompt triggering an adequate output as a basis for an in-depth analysis. We defined the following conditions as a starting point for the creation of the prompt. Some of these are based on Denny et al.'s (Denny et al. 2021) guidelines for improving the readability of programming error messages.

- The feedback should be concise and not too lengthy. This condition relates to the 'economy of words' guideline, which means that 'only as many words are used as are necessary to communicate the point' (Denny et al. 2021).
- The feedback should be targeted at novice programmers. We assume we have no further information on the learner, such as skill level, problem-solving history, or other context. This condition relates to the 'remove jargon' and 'use simple vocabulary' guidelines (Denny et al. 2021).
- Although feedback can be defined on many levels (high-level vs. bottom-up hints), the feedback should aim at giving informative information and/or hints, but not give away the solution.
- The generated feedback should be only one of the specific types described in Section 2.1.
- We have access to the student's code, the task description and if applicable/available template (starter) code and test cases.

Denny et al.'s (Denny et al. 2021) fourth and final guideline 'write messages in complete sentences' also applies, as LLMs typically output complete sentences.

To refine the prompt, we followed the steps below:

- Using the existing version of the prompt, we generated feedback for the three student submissions within the test set across all six types of feedback, resulting in a total of 18 feedback messages for each iteration.
- Two authors analysed the feedback messages, by indicating which feedback types were present in the output and determining the characteristics. The results were discussed by the three authors of this article. Disagreements were discussed and resolved using a consensual approach (Hill, Thompson, and Williams 1997; Hill et al. 2005).
- To determine whether the prompt should be further refined, we checked whether the expected feedback type was present, if there was misleading information or uncertainty. We then discussed improvements to the prompt and make adjustments accordingly.

We selected OpenAI's GPT-4 model (OpenAI 2023) to generate the outputs, as one of the state-of-the-art LLMs at the time the study was conducted (March 2024). Even though its price might be a barrier for educational institutions and students, we expect this model soon to be standard (and potentially free of charge). The ChatGPT interface was utilised to generate the feedback, while a separate context window was used for each request. In the following paragraphs, we summarize the five iterations of the prompt design process by outlining crucial changes.

TABLE 4 | Overview of selected student submissions.

ID	Dataset	Exercise	Language	Errors
Test set				
10_TEOS	FIE	Physics	Java	*Syntax errors*
				Use of uninitialised variable
				Missing variable type
13_NEGF	FIE	NegaFib	Java	*Semantic errors*
				Wrong multiplication operator
				Incorrect parameters of recursive function call
				other:
				Forbidden library usage
01_TTBSA	FIE	Triangles	Java	*Semantic errors*
				Missing checks for illegal triangles
				Unnecessary checks
Added programs for complete evaluation set				
FIB_01	Codingbat	Fibonacci	Java	*Semantic errors*
				Wrong recursive call
FIB_02	Codingbat	Fibonacci	Java	*No Errors*
PASS	FITech	Password	Dart	*Semantic errors*
				Overwriting variable
				Incorrect logic (missing loop)
				Wrong datatype
MAX3_01	TaskTracker	MaxOf3	Python	*Semantic error*
				Incorrect conditional
MAX3_02	TaskTracker	MaxOf3	Kotlin	*No Errors*
MAX3_03	TaskTracker	MaxOf3	C++	*Style issues*
				Unusual var. names (due to post-processing)
				Unnecessary import
MAX3_04	TaskTracker	MaxOf3	Java	*No Errors*
MAX3_05	TaskTracker	MaxOf3	Java	*Semantic errors*
				Missing conditionals
				Unusual approach using subtraction in conditions

3.4.1 | First Iteration

We designed a first prompt based on our initial conditions, using best practices described in documents and literature (Zamfirescu-Pereira et al. 2023; OpenAI 2024). We included the following elements:

- A system prompt with instructions about the target audience, the elements given below, and an explanation of the different feedback types.
- The task description, translated from German into English.
- The class skeleton given to the student.
- The student's submitted solution.
- A model solution.
- The type of feedback we want the model to generate (KR, KP, KC, KTC, KM or KH).

When analyzing the output, we noticed several issues: First, the output was too long for the KR feedback (290 words on average), for which we only expected a simple correct/incorrect statement with no more than five words (e.g., *'The solution*

is incorrect'). In multiple instances, the response contained further feedback types in addition to the requested feedback type. Another critical issue was that the output referred to the model solution, and kept comparing the input to the model solution. This is a common problem that has been observed in previous studies (Roest, Keuning, and Jeuring 2024; Hellas et al. 2023).

To mitigate these issues, we added instructions to have only correct/incorrect as output for KR (see line 18 in Listings 1), and to not provide any additional information than the desired feedback type (DFT) (line 24). We also instructed the model that the feedback should never contain information about the model solution (line 25).

3.4.2 | Second Iteration

We observed several improvements within the second iteration: the output was shorter, KR feedback only gave information about correct/incorrect, and it seemed that the feedback was more related to the requested feedback types. We did not see any mentions of the model solution anymore. In terms of issues, KTC responses sometimes just repeated the task requirements. KH was in some cases not precise, only saying '*check if correct*'. KC gives information about concepts even if there is no issue in the code relating to that concept. KP feedback was in some cases incorrect, vague, or only KR feedback.

We made some substantial changes to our prompt. For KC, we instructed the model to focus only on programming concepts and explain these concepts only if the issue in the submitted solution of the student is related to that concept (see line 18 in Listing 1). For KP, we explained that we want the model to try to divide the task into useful subgoals and give feedback about the current state of the student submission as a percentage (see line 19). For KH feedback, we wanted the model to not only generate an instruction for the student to check if something is correct but also provide hints and suggestions on how to proceed (see line 20). For KTC, we instructed the model to only point to information in the task description if there is an issue in the student submission related to that task constraint (see line 21).

3.4.3 | Third Iteration

In the output of the third iteration, we noticed that KM feedback also gave hints on how to proceed, which we do not want. Moreover, the responses had become lengthier again, using verbose language which might not be appealing to novices. There were never any (code) examples or motivational words. In addition, the given type of feedback resembled our instruction. KP gave a lengthier description instead of just a percentage, which we decided to keep so it is clear what it is based on.

In our improvements, we simplified the definition supplied with KR about when a solution is considered correct to be either '(1) programming solution passes all tests and contains no mistakes (2) programming solution is semantically equivalent to the model solution.' We also told the model that the feedback has to be brief and precise. For KM feedback, we wanted to give the student information about the error and not show hints on how to fix it. We also tried to experiment with adding illustrating examples to better support students' understanding, which we requested for KM, KTC, KC and KH.

3.4.4 | Fourth Iteration

The results of our previous changes did not seem to significantly improve the output. Firstly, we noticed lengthy responses. The model clearly tried to generate examples, however, we observed that the term 'example' can be used in many different ways. To illustrate, the model just repeated some code from the student with issues, to be used as an 'example' for a mistake. We noticed a few interesting and useful examples, such as '*For example, if* '$a = 1$', '$b = 2$', *and* '$c = 3$', *your current implementation would classify this as a SCALENE triangle, which is incorrect because these sides do not meet the triangle inequality theorem and thus cannot form a triangle*'. We also noticed more code examples, but they were not always compilable. This happened when the model suggested replacing the constants defined by the student with constants from a given class:

```
For example, in your computeP method, you
directly defined BOLTZMANN and AVOGADRO con-
stants:
double BOLTZMANN=1.380649E-23;
double AVOGADRO=6.02214076E23;
Instead, you should utilize the constants
from the \texttt{PhysicsConstants} class like
so:
PhysicsConstants.BOLTZMANN
PhysicsConstants.AVOGADRO
```

We reworded the instruction to only show the errors for KM and not how to fix it, and we removed the instruction to give examples. At this point, we found we could not control the example output enough, and realised that examples require more extensive attention at some point.

3.4.5 | Fifth Iteration and Final Prompt

In the fifth iteration, we successfully addressed the issues encountered in the fourth iteration. Yet, we observed no notable enhancements compared with the third iteration. Consequently, we chose to use this version of the prompt to assess the expanded dataset. Our final prompt is shown in Listing 1.

3.5 | Feedback Analysis

To answer RQ1 and RQ2, we analysed the generated feedback with regard to their characteristics and the actual feedback type(s) (AFT), as described in Section 2.1. For the characteristics (see Table 2), we used the deductive categories from previous work (Kiesler, Lohr, and Keuning 2023), extended by the two categories PERS (Azaiz, Kiesler, and Strickroth 2024; Azaiz, Deckarm, and Strickroth 2023; Roest, Keuning, and Jeuring 2024) and COMPL taken from other related work

LISTING 1 | Final prompt used to generate different types of feedback.

```
Your task is to give feedback to a novice programmer who is working on a specific pro-
gramming exercise in Java. You will be given (A) the complete task description as a LaTex
Source Code, (B) the class body that is provided to the student, (C) the submitted solu-
tion of the student, (D) a model solution and (E) a desired feedback type with regard to
the feedback typology described below.
Each feedback type is described with the name of the feedback type in wrapped in "" fol-
lowed by a description of the feedback type wrapped in ~~
"Knowledge about task constraints (KTC)" ~ KTC are elaborated components that provide in-
formation of task rules, task constraints, and task requirements ~
"Knowledge about concepts (KC)" ~ KC are elaborated components that provide information on
conceptual knowledge relevant for task processing ~
"Knowledge about mistakes (KM)" ~ KM are elaborated components that provide information on
errors or mistakes ~
"Knowledge on how to proceed (KH)" ~ KH are elaborated components that provide information
on procedural knowledge relevant for task processing ~
"Knowledge about performance (KP)" ~ KP provides summative feedback on the achieved per-
formance level after doing multiple tasks or subtasks, such as '15 of 20 correct' and '85%
correct ' ~
"Knowledge of result/response (KR)" ~ KR is feedback that communicates whether a solution
is correct or incorrect e.g. (1) programming solution passes all tests and contains no
mistakes (2) programming solution is semantically equivalent to the model solution ~
Try to make sure that the feedback you generate satisfies the following criteria:
- for feedback type KR the feedback should only contain the information if the student
submission is correct or incorrect. Do not explain concepts or give information about er-
rors or how to proceed.
- for feedback type KC only focus on programming concepts and explain these concepts only
if the issue in the submitted solution of the student is related to that concept.
- for feedback type KP try to divide the task into usefull subgoals and give feedback
about the current state of the student submission in percentage.
- for feedback type KH not only tell the student to check if something is correct but pro-
vide hints and suggestions on how to proceed.
- for feedback type KTC only point to information in the task description if there is an
issue in the student submission related to that task constraint.
- for KM Feedback just give the student information about the error and do not describe
how to fix the error.
- make sure that the generated feedback corresponds exactly to the description of the de-
sired feedback type. Do not provide any additional information that does not match the
feedback type.
- the feedback should never contain information about the model solution. The student does
not know the model solution.
- your feedback has to be brief and precise.
(A) complete task description:
## INSERT Add task description here ##
(B) class body in file "##INPUT## Name of File" provided to the student:
## INSERT Add Class Body, if provided ##
(C) student's submitted solution:
## INSERT Add Student Solution ##
(D) model solution:
## INSERT Model solution ##
(E) desired feedback type:
## INSERT desired feedback type ##
```

(Azaiz, Kiesler, and Strickroth 2024; Azaiz, Deckarm, and Strickroth 2023). The PERS characteristic indicates that the feedback is personalised, meaning it explicitly refers to the student's solution. The COMPL characteristic addresses whether the feedback complies with the task description.

The analysis was carried out in three iterations with two steps each: (1) two experts (independently) coded the generated feedback messages with the appropriate feedback types and categories, and (2) all conflicts were discussed and resolved using the consensual approach (Hill, Thompson, and Williams 1997; Hill

et al. 2005). In order not to be biased when coding the generated feedback, we concealed the information on which feedback type was requested and by randomly ordering the messages. Finally, the coded data were qualitatively analysed by identifying representative themes in the feedback, and illustrative examples.

4 | Results and Discussion

4.1 | RQ1: Generating Specific Feedback Types

The results of the analysis of the generated feedback types are summarised in Table 5. It shows that in almost all cases (63 out of 66), the DFT matches the AFT generated by the LLM. However, in 19 cases, other feedback types were additionally included in the generated feedback. It was particularly noticeable that KM feedback was often included when requesting KH feedback. However, this may be because it is not always possible to provide information on how to proceed without (directly or indirectly) pointing to the problem in the code. The same effect was observed when requesting KTC feedback. In some cases, KTC feedback was also combined with KM or KH feedback, as it is challenging to identify problems regarding the task constraints without giving any indication of the error in the code. Regarding RQ1, we conclude that the performance of the GPT-4 model to generate specific, elaborated types of feedback is very promising.

4.2 | RQ2: Characteristics of Generated Feedback

In this section, we describe the characteristics we found in the generated feedback messages and conduct a qualitative analysis in which we highlight certain themes identified in the data.

4.2.1 | Dealing with Correct Programs

Several correct student programs were used as input to the LLM to find out the extent to which GPT-4 can recognize these and provide adequate feedback. For example, consider the student program in Listing 2, which is a correct `Kotlin` solution to the MaxOf3 exercise. The test cases for this exercise were shown in the form of text as '1 2 3 → 3'. However, the submissions required the input to be inserted line-by-line. The KTC feedback confuses these by stating '*... your current implementation reads three integers in a single line, which may not align with the test cases provided in the task description where inputs are expected line by line*', with the advice to '*review the task description and adjust your input reading mechanism to match the expected format of the test cases*'. Therefore, we consider this feedback to be misleading. We observed similar misleading feedback for a correct `C++` and a correct `Java` submission to the same task. For the `C++` program, the LLM responded: '*Your program should output only the largest number, without any preceding text*'. However, the student had already done this and only printed the largest number (which was also the case in the `Java` submission).

The other feedback types for the `Kotlin` submission (Listing 2) all contain advice on style. It is mentioned that the code could be made '*cleaner*' and more idiomatic to `Kotlin` by using `readLine()`. This is perfectly fine advice; however, in the KM feedback the term 'mistake' is used for this issue, which is misleading since it is not really problematic. This is also the case for the KP feedback, which awards a score of 85% for this correct program that would not need changes.

The KC feedback for a correct `C++` submission gave confusing feedback on the student using the wrong header file for the built-in max-function, while this import was not even necessary. The KM feedback for the same solution acknowledges this issue as well, but phrases it as a style suggestion: '*Removing unused libraries helps to keep the codebase more maintainable and potentially reduces the compilation time.*'

For a correct but somewhat overly complex solution to MaxOf3 in `Java`, GPT-4 suggested useful improvements: '*consider structuring your conditional statements to more clearly identify the largest number among the three inputs. Start by comparing the first number with the second and third numbers to determine if it's the largest. If it's not, move on to compare the second number with the third. This way, you can avoid redundant comparisons and make your logic more straightforward.*'

4.2.2 | Misleading Feedback

One of the major issues pointed out by previous studies is the regular occurrence of misleading advice. We still noticed such issues in the feedback, but misleading information seemed to occur less frequently (compared with (Kiesler, Lohr, and Keuning 2023)). One example of misleading feedback pointed out '*a minor inconsistency in the use of braces in the "if-else" blocks, where one of the "else" statements does not use braces for a single statement, unlike the others*'. This was not the case in the actual student code. Therefore, it may be very confusing feedback for a novice learner. In the next sections, we highlight examples of misleading feedback for specific feedback types.

4.2.3 | Feedback on Overall Performance (KP)

Although our focus was on formative feedback, we were also interested in getting feedback on performance in general. While KP is usually considered a simple feedback type, with a number or percentage indicating progress, we decided to generate more elaborate reports on student's progress. We used the concept of 'subgoals' (Margulieux, Catrambone, and Guzdial 2016), which has been studied extensively in computing education research. The results for this feedback type were quite good, although we did not always agree with the percentage score given by the model, often giving too many points for minor subgoals.

For the MaxOf3 exercise, GPT-4 divided the problem into three reasonable subgoals in most cases, for example, '(1) *reading inputs correctly*, (2) *comparing the numbers accurately, and* (3) *printing the correct output.*' For most correct submissions, the expected score of 100% was given, together with a description of how the solution meets the subgoals. However, we also spotted an instance where a score of 80% was displayed for a correct solution.

TABLE 5 | How ChatGPT responds to student input (RQ1). For legend see Table 2. DFT is desired feedback type, AFT is actual feedback type.

	FB types		Content						Quality			Other		
Stud_task	DFT	AFT	INFO	STYLE	CAUSE	FIX	CODE	EXA	COMP	MIS	UNC	META	MOT PERS	COMPL
10 TEOS	KR	KR	○	○	○	○	○	○	—	○	○	○	○ ●	○
	KTC	KTC	○	○	●	●	○	○	—	○	○	○	○ ●	●
	KC	KC KM	○	○	●	●	○	○	—	○	○	○	○ ●	●
	KM	KM	○	○	●	●	○	○	—	○	○	○	○ ●	●
	KH	KH	○	○	●	●	○	○	—	○	○	○	○ ●	●
	KP	KP	○	○	●	○	○	○	—	○	○	○	○ ●	●
13 NEGF	KR	KR	○	○	○	○	○	○	—	○	○	○	○ ●	○
	KTC	KTC KM	○	○	●	●	○	○	—	○	○	○	○ ●	●
	KC	KC KM	○	○	●	●	◐	○	—	○	○	○	○ ●	●
	KM	KM	○	○	●	●	◐	○	—	○	○	○	○ ●	●
	KH	KH KM	○	○	●	●	◐	○	—	○	○	○	○ ●	●
	KP	KP KM	○	○	●	●	○	○	—	○	○	○	○ ●	●
01 TTBSA	KR	KR	○	○	○	○	○	○	—	○	○	○	○ ●	○
	KTC	KTC	○	○	●	●	○	○	—	○	○	○	○ ●	●
	KC	KC KM	○	○	●	●	○	○	—	○	○	○	○ ●	●
	KM	KM	○	○	●	●	○	○	—	○	○	○	○ ●	●
	KH	KH	○	○	●	●	○	○	—	○	○	○	○ ●	●
	KP	KP KM	○	○	●	○	○	○	—	○	○	○	○ ●	●
FIB 01	KR	KR	○	○	○	○	○	○	—	○	○	○	○ ●	○
	KTC	KTC	○	○	●	○	◐	○	—	○	○	○	○ ●	●
	KC	KC KM	○	○	●	○	○	○	—	○	○	○	○ ●	●
	KM	KM KH	○	○	●	●	◐	○	—	○	○	○	○ ●	●
	KH	KH KM	○	○	●	●	◐	○	—	○	○	○	○ ●	●
	KP	KP KM	○	○	●	○	◐	○	—	○	○	○	● ●	●

(Continues)

TABLE 5 | (Continued)

	FB types		Content						Quality			Other		
Stud_task	**DFT**	**AFT**	**INFO**	**STYLE**	**CAUSE**	**FIX**	**CODE**	**EXA**	**COMP**	**MIS**	**UNC**	**META**	**MOT PERS**	**COMPL**
FIB 02 ✓	KR	KR	○	○	○	○	○	○	—	○	○	○	○ ●	○
	KTC	KTC	○	○	○	○	○	○	—	○	○	○	○ ○	●
	KC	KC	○	○	○	○	○	○	—	○	○	○	○ ●	●
	KM	KM KH	○	○	●	●	◐	○	—	●	○	○	○ ●	●
	KH	KH KC	○	○	○	○	○	○	—	○	○	○	○ ○	○
	KP	KP KM	○	○	○	○	○	○	—	○	○	○	○ ●	●
PASS 01	KR	KR	○	○	○	○	○	○	—	○	○	○	○ ●	○
	KTC	KTC KM KH	○	○	●	●	○	○	—	○	○	○	○ ●	●
	KC	KC, KM	○	○	●	○	○	○	—	○	○	○	○ ●	●
	KM	KM	○	○	●	○	◐	○	—	○	○	○	○ ●	●
	KH	KH	○	○	○	●	○	○	—	○	○	○	○ ●	●
	KP	KP KM	○	○	●	○	○	○	—	○	○	○	○ ●	●
MAX3 01	KR	KR	○	○	○	○	○	○	—	●	○	○	○ ●	○
	KTC	KTC	○	○	○	○	○	○	—	●	○	○	○ ●	●
	KC	KC KM	○	○	●	●	○	○	—	○	○	○	○ ●	●
	KM	KM	○	○	●	○	◐	●	—	●	○	○	○ ●	●
	KH	KH	○	●	○	●	○	○	—	○	○	○	○ ●	●
	KP	KP KM	○	○	○	●	○	○	—	●	○	○	○ ●	●
MAX3 02 ✓	KR	KR	○	○	○	○	○	○	—	○	○	○	○ ●	○
	KTC	KTC	○	○	●	●	○	○	—	●	○	○	○ ●	○
	KC	KC	○	●	○	○	◐	○	—	○	○	○	○ ●	●
	KM	KM KH	○	●	●	●	◐	○	—	●	○	○	○ ●	●
	KH	KH	○	●	○	●	○	○	—	●	○	○	○ ●	●
	KP	KP KM	○	○	○	○	○	○	—	●	○	○	○ ●	●

(Continues)

TABLE 5 | (Continued)

	FB types		Content						Quality			Other		
Stud_task	**DFT**	**AFT**	**INFO**	**STYLE**	**CAUSE**	**FIX**	**CODE**	**EXA**	**COMP**	**MIS**	**UNC**	**META**	**MOT PERS**	**COMPL**
MAX3 03 ✓	KR	KR	○	○	○	○	○	○	—	○	○	○	○ ●	○
	KTC	KTC	○	○	○	○	○	○	—	●	○	○	○ ●	●
	KC	KC	○	○	○	○	○	○	—	○	○	○	○ ●	●
	KM	KM KH	○	●	●	●	○	○	—	○	○	○	○ ●	●
	KH	KH	○	●	●	●	○	○	—	●	○	○	○ ●	●
	KP	KP KM	○	○	○	○	○	○	—	●	○	○	○ ●	●
MAX3 04 ✓	KR	KR	○	○	○	○	○	○	—	○	○	○	○ ●	○
	KTC	KTC	○	○	●	○	○	○	—	●	○	○	○ ●	●
	KC	KC	○	○	○	○	○	○	—	○	○	○	○ ●	●
	KM	KM	○	●	●	○	○	○	—	?	○	○	○ ●	●
	KH	KH	○	●	○	●	○	○	—	○	○	○	○ ●	●
	KP	KP	○	○	○	○	○	○	—	○	○	○	○ ●	●
MAX3 05	KR	KR	○	○	○	○	○	○	—	○	○	○	○ ●	○
	KTC	KM KH	○	○	●	●	○	○	—	○	○	○	○ ●	●
	KC	KC	○	●	○	○	◐	○	—	○	○	○	○ ●	●
	KM	KM	○	○	●	○	○	○	—	○	○	○	○ ●	●
	KH	KH KM	○	○	●	●	○	○	—	○	○	●	○ ●	●
	KP	KP KM	○	○	○	○	○	○	—	○	○	○	○ ●	●

Note: ●•: Yes ○: No ◐: Snippet —: Not applicable ✓: Correct Solution.

LISTING 2 | Correct Kotlin submission to MaxOf3.

```
 1 import java.util.*
 2 import kotlin.math.max
 3
 4 fun main() {
 5    val main_v1 = Scanner(System.`in`)
 6    val main_v2 = main_v1.nextInt()
 7    val main_v3 = main_v1.nextInt()
 8    val main_v4 = main_v1.nextInt()
 9    println(max(max(main_v2, main_v3),
main_v4))
10 }
```

4.2.4 | Feedback on Concepts (KC)

We instructed the model to focus on *programming* concepts, as opposed to other concepts related to the task description, such as mathematical formulas. We expected these concepts to appear in feedback on (semantic) mistakes. In most cases, this worked out well. For the correct solutions, GPT-4 usually made general remarks about a concept, possibly referring to how it is applied in the student's solution. For the incorrect solutions, the concepts were related to actual mistakes and, in some cases, contained some information on how to fix them.

As an example, for an incorrect solution to the NegaFib exercise, the output for KC focused on the concept of recursion, giving rather a long explanation, including '*each function call should ideally break down the problem into smaller, more manageable subproblems that resemble the original problem*'. This feedback also contained some CAUSE and FIX elements, to point out the mistake made related to recursion.

For a `Kotlin` solution to MaxOf3, GPT-4 addressed the concept of the library function `max`: '*You've nested two* `max` *functions to compare three values:* `max(max(main_v2, main_v3), main_v4)`. *This is a valid approach to find the largest of three numbers...*'. In addition, it stated *'However, it's important to understand that Kotlin also provides conditional expressions, such as if statements and when expressions, that can be used to solve this problem in a more readable or structured way, especially when dealing with more complex conditions or a larger set of numbers'*. Other concepts that GPT-4 brought up were `C++` header files, comparison operators, constants, and variable scope.

Another subtype of KC deals with giving examples. None of the generated feedback messages contained complete code examples. This is hardly surprising, as we explicitly requested in our prompt that no solution should be provided. However, code snippets were generated in many explanations (especially for types KM, KH and KTC). These consisted, for example, of references to parts of the code. As there were no complete code examples in the feedback messages, the 'COMP' category was obsolete. It was noticeable that there were hardly any examples in the feedback messages. By examples, we mean illustrative examples that, for example, underpin the explanation of a concept or make it easier to understand. We explicitly did not code references to parts of the code as examples.

4.2.5 | Feedback on Task Constraints (KTC)

This type of feedback encompasses hints on task requirements, such as enforcing the use of a particular language construct or prohibiting something. Another subcategory contains hints with general information on how to approach the exercise, not being specific to the student's current work. In some of the outputs we noticed GPT-4 giving feedback of this kind, such as for the Physics task: '*The task specifically requires you to use the constants defined in the PhysicsConstants class for these values to ensure consistency and maintainability of the code*.', and the NegaFibonacci: '*The task explicitly requires the use of the generalized Fibonacci sequence formula for negaFibonacci, without resorting to alternative formulas or approaches*.'

In other cases, the feedback referred to *functional* constraints that can be derived from the task description, which we assign to KM feedback on solution errors. For the correct solutions to the MaxOf3 exercise, which did not have any particular constraints, the generated feedback mentioned something about checking the output format. It even gave misleading suggestions that *'Your program should output only the largest number, without any preceding text'*, while this was not the case. In several cases, we also noticed that GPT-4 simply rephrased the task description.

4.2.6 | Feedback on Mistakes (KM)

There are several subtypes to this category: feedback on compiler errors, solution errors, test failures, style issues and performance issues. This feedback can be short, but we mostly observed elaborate error descriptions. We instructed GPT not to give information on how to fix the error, but in some cases, this was still included. In most cases, GPT pointed out actual errors but made a few mistakes. For example, for the program below in Listing 3 the feedback was *'Your solution correctly implements the Fibonacci sequence; however, there's a mistake in the order of your base case checks. You should check for n == 0 before n == 1 to adhere to the sequence's definition starting from 0'*. This might indeed look somewhat nicer but does not have any consequences for the method's correctness.

In general, we observed more often that GPT used the term 'mistake' for issues related to style, which did not influence the correctness of the code. We consider this to be misleading for students, because they will think they did something wrong, instead of just giving a hint for an improvement. This is probably because we call this type 'mistake', but we actually mean it in a broader sense (e.g., to include style issues).

KM inherently always contains a CAUSE explaining where or what the error is and is therefore also personalised (PERS). In some cases, style suggestions were given. It was noticeable that this was particularly prevalent in feedback messages on correct tasks.

LISTING 3 | Correct Java submission to Fibonacci (FIB02).

```
1 public int fibonacci(int n) {
2   if(n==1) return 1;
3   if(n==0) return 0;
4   return fibonacci(n-2) + fibonac-
ci(n-1);
5 }
```

LISTING 4 | Fragment of incorrect Java submission to MaxOf3 (omitting initialization of variables and scanner.

```
 1 //..
 2 main_v1 = main_v7.nextInt();
 3 main_v2 = main_v7.nextInt();
 4 main_v3 = main_v7.nextInt();
 5 main_v4 = main_v1 - main_v2;
 6 if (main_v4 <=0){
 7   if ((main_v2 - main_v3)<=0){
 8     System.out.println(main_v3);
 9   } else {
10     System.out.println(main_v2);
11   }
12 }
```

4.2.7 | Feedback on How to Proceed (KH)

The CAUSE and FIX categories are closely linked to the KM and KH feedback types. If KM feedback was requested, CAUSE was always included. The same applies to KH feedback. This is not surprising, as Kiesler et al. (Kiesler, Lohr, and Keuning 2023) had derived these two categories from KM and KH feedback. In many cases, the KH feedback gave valuable instructions—formulated as individual steps—on how to address the problems in the code. In some cases, the feedback was less valuable as it contained information that in principle corresponds to a 'think harder' statement, for example, simply asking the learner to ensure that a particular specification was correct without concrete help on how to achieve this ('*Ensure your method correctly implements the formula provided in the task description*'.).

4.2.8 | Dealing with Logic

It is well known that LLMs struggle with logical reasoning (Creswell, Shanahan, and Higgins 2023). In one `Java` solution to MaxOf3 the student had the unusual approach of comparing numbers by subtracting them and checking for a positive or negative result (see Listing 4). GPT-4 was not always precise in pointing out the problem, noting that *'your solution currently lacks a branch to handle the situation where the first number is less than the second but greater than the third'*, which is not a particular case that needs to be dealt with. Moreover, the model added that *'adding this condition will make your solution more comprehensive'*, which does not seem like a useful goal.

4.2.9 | Language and Tone of the Feedback

In our prompt, we explicitly instruct ChatGPT to give feedback to a *novice* programmer. In addition, our exercises are clearly aimed at beginners. We noticed that the generated feedback was suitable for the target audience. However, there were some instances of specific terminologies, such as *'The submitted solution has an error related to variable shadowing and data type mismatch'*, but in this case, the errors were explained afterward.

As in a previous study (Kiesler, Lohr, and Keuning 2023), very few feedback messages contained motivational components. Another study (Roest, Keuning, and Jeuring 2024) also identified few compliments in feedback messages, although observed mostly a 'friendly' tone. We only coded the MOT category if there was a clear intention to motivate (e.g., 'Well done'). The feedback sometimes provided information about (partial) correct solutions (e.g., 'Your implementation of the base cases is correct'). While these cases may serve the purpose of motivating the learner, we did not label them as MOT because we could not determine the 'real intention' behind the generated messages. The only motivating text we noticed was '*Your current solution has made a good attempt at implementing the Fibonacci sequence, but it's not fully aligned with the requirements of the Fibonacci calculation...*'

Kiesler et al. (Kiesler, Lohr, and Keuning 2023) also reported feedback messages asking for more information or showing uncertainty when providing feedback to learners' submissions. In contrast, we did not observe any feedback messages requesting additional information or responses with uncertainty. This may be attributed to the more comprehensive context provided in the prompt during generation. We do not assume that this improvement is because we used a newer model (GPT-4 instead of GPT-3.5).

Fortunately, we did not find any instances of derogatory or demeaning tone in the generated feedback messages. The language was consistently grammatically correct and expressed in a friendly or neutral tone.

5 | Limitations

The results presented in this article are based on the evaluation of outputs from a single LLM (GPT-4). The evaluation of such feedback by experts is a time-consuming task, so we had to decide to either (a) evaluate multiple models on a small set of different tasks or (b) evaluate a more heterogeneous set of tasks and responses from a single model. We chose the second option as the GPT-4 model seemed to be the most promising and investigated model (at the time of the study). In addition, we assume it is more valuable to investigate multiple programming languages, topics and error types, and how one system responds to them.

Another limitation is due to the selected programming exercises as part of our dataset, which mostly concern mathematical functions. Although this is common in the first few weeks of

introductory programming courses, it does not cover all possible types of tasks. Therefore, we have to be careful to generalize our results to other (non-mathematical) task domains in introductory programming.

6 | Implications and Recommendations

6.1 | Implications for Educators

Incorporating LLMs for programming in educational settings can be beneficial for improving the feedback loop, alleviating teachers from frequently having to provide individual feedback. The potential effectiveness of LLM-driven feedback, however, highly depends on the precision and contextualization of the used prompt. Previous research has highlighted the limitations of overly simplistic prompts. Simply providing student code with a general request for feedback may lead to (partially) incorrect or misleading responses, or the model requesting more information (Kiesler, Lohr, and Keuning 2023). The present study reveals that it is possible to generate specific feedback types tailored to the investigated tasks.

Consequently, we can assume two use cases for (online) course scenarios: (1) The educator selects a suitable feedback type and submits the respective prompt(s) to the model. (2) Learners make their requests to the model to generate specific feedback types (e.g., during the programming process). In both cases, some guidance or introduction is needed on what to consider when requesting feedback from the model. Studies have shown that requests to the model without a certain level of basic understanding lead to poorer results (Prather et al. 2023; Jeuring, Groot, and Keuning 2024). If students can use predefined prompt(s) (i.e., those investigated in this article), students can simply select the DFT. For example, if they only want a hint on how to proceed to correct the faulty code (KH) or information about the error and where it is located in the code (KM). Or does the learner want a report on their progress of the task (KP)? The choice would be up to them. Instead of requiring learners to adapt the prompt themselves, the feedback request should be facilitated through a simplified method, such as a button sending the appropriate prompt to request and display the desired feedback, avoiding the need for direct interaction of the learner with the prompt (see (Lohr et al. 2024)). Unfortunately, the LLM still occasionally hallucinates and produces misleading information, which is a major concern—especially if the output is merely available to the learners and not reflected or critically discussed. Learners need to become aware that the feedback they receive may sound good but actually contains misleading information. Educators should, therefore, support students in that learning process.

6.2 | Implications for Tool Developers

The results of this study have shown that the generation of feedback for programming tasks using LLMs has the potential to improve educational systems. The major advantage over previous approaches is that LLMs do not have to be trained by users themselves on large amounts of data, which saves considerable development efforts. Nevertheless, we believe an approach based solely on LLMs is not the most promising. Our results show that even simple feedback such as KR is not always correct, claiming that a correct submission was incorrect, and vice versa. Hybrid solutions may be more promising, in which, for example, test cases are combined with requests to the LLM. Then test cases are evaluated and their outcomes are inserted into the prompt as information. We assume that in this case, the probability of misleading information in KR is significantly lower and the KP feedback will also be of higher quality. We have seen that in the case of (correctly diagnosed) solutions, the feedback often contained hints on how to improve the code in terms of readability and structure (style). A hybrid approach with test cases and LLMs could therefore create further opportunities to adapt the feedback even more specifically to the status of the current submission—for example, *'Give only STYLE suggestions iff the submitted solution passes all tests'*.

6.3 | Implications for Researchers

In this study, we have created and analysed a dataset consisting of LLM-generated feedback on authentic student submissions in an introductory programming course. Most of the feedback corresponded to the feedback types we intended to generate, but we only examined the quality of the feedback by looking at certain characteristics. In future study, researchers could investigate students' and experts' perspectives on the helpfulness of the generated feedback. So far, we generated one type of feedback for each request. In a real-world scenario, feedback from experts does not exclusively consist of only one type. It is therefore worth investigating whether and how several types of feedback can be combined and generated in conjunction. In terms of expanding the feedback variants, researchers can investigate how to generate even more specific types of feedback, such as the subtypes from the applied feedback classification (Keuning, Jeuring, and Heeren 2018; Narciss 2006). Another pathway is to investigate different levels of detail, from high-level hints to concrete suggestions, and to personalize the feedback even more to students' preferences. The provided prompt could also be adapted for more complex tasks.

7 | Conclusion

In this study, we investigated how LLMs could be instructed to generate feedback of different types (according to an existing feedback classification) for solutions to introductory programming exercises. We incrementally developed a prompt in five iterations, arriving at a version that generated the requested feedback types in most cases. We analysed the output of the final prompt using GPT-4 for each of the six feedback types and 11 student solutions. The student submissions for a total of six programming exercises contain various kinds of errors and are written in four different programming languages (`Java`, `Python`, `C++` and `Kotlin`). We qualitatively analysed the resulting 66 feedback messages by categorizing the resulting feedback type(s) and recording the 13 characteristics defined in previous study, resulting in the analysis of 858 features.

We found that our prompt generated the DFT in most of the cases (63 of 66). At the same time, we identified elements that

were not requested. The prompt for KR feedback always provided only a correct/incorrect evaluation, which in most cases matched the actual (in)correctness of the solution. The KTC feedback prompt mostly generated relevant task constraints, such as the absence of a required concept, and to give feedback accordingly. However, we noticed that functional constraints were generated, which we would have expected as part of KM feedback. The KC feedback prompt introduced programming concepts identified in the exercises students struggled with. It was noted in combination with KM elements. In general, the KM feedback was very good at pointing out the errors in student programs, showing an improvement in earlier studies with older models. In some cases, the fixes to these problems were also included (KH). Finally, we observed that GPT-4 was good at generating KP feedback by dividing the problem into subgoals. Nonetheless, the resulting percentages indicating correctness did not always match our estimations.

Earlier study has shown that existing learning environments lack diversity in the types of feedback they provide, usually resorting to types that are easy to automate, for example, by showing test case results. With this study, we propose the prompting of LLMs in a controlled way, to give richer feedback to students not limited to scores and lists of mistakes. LLMs, such as GPT-4, can generate explanations for relevant programming concepts, point to task requirements, hints on how to address mistakes and information about students' progress throughout the task. This way, educators and learners are enabled to apply LLMs in a way that can be beneficial, providing appropriate guardrails, as opposed to giving them the freedom to request a solution to the task by asking ChatGPT and causing over-reliance on GenAI tools (Prather et al. 2023). We further described implications for educators, researchers and tool developers, advocating for a hybrid approach to combine LLM-based feedback with established methods and human instruction. In addition, this research will support researchers and teachers with the means to better study the effects of different types of feedback in the future.

Author Contributions

Dominic Lohr: conceptualization, methodology, data curation, investigation, validation, formal analysis, writing – original draft. **Hieke Keuning:** writing – original draft, conceptualization, methodology, data curation, supervision, formal analysis, investigation. **Natalie Kiesler:** writing – original draft, conceptualization, methodology, data curation, supervision, investigation.

Conflicts of Interest

The authors declare no conflicts of interest.

Data Availability Statement

The data that support the findings of this study are available on request from the corresponding author. The data are not publicly available due to privacy or ethical restrictions.

References

Azaiz, I., O. Deckarm, and S. Strickroth. 2023. "AI-enhanced Auto-Correction of Programming Exercises: How Effective is GPT-3.5?" *International Journal of Engineering Pedagogy (iJEP)* 13, no. 8: 67–83.

Azaiz, I., N. Kiesler, and S. Strickroth. 2024. "Feedback-Generation for Programming Exercises With GPT-4." In *Proceedings of the 2024 on Innovation and Technology in Computer Science Education V. 1 ITiCSE 2024*, 31–37. New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/3649217.3653594.

Balse, R., B. Valaboju, S. Singhal, J. M. Warriem, and P. Prasad. 2023. "Investigating the Potential of GPT-3 in Providing Feedback for Programming Assessments." In *Proceedings of the 2023 Conference on Innovation and Technology in Computer Science Education V. 1 ITiCSE 2023*, 292–298. New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/3587102.3588852.

Bayman, P., and R. E. Mayer. 1988. "Using Conceptual Models to Teach BASIC Computer Programming." *Journal of Educational Psychology* 80, no. 3: 291–298.

Becker, B. A., M. Craig, P. Denny, et al. 2024. "Generative AI in Introductory Programming." In *Computer Science Curricula 2023 New York*, edited by A. N. Kumar, R. K. Raj, S. G. Aly, et al., 438–439. NY, USA: Association for Computing Machinery. https://csed.acm.org/wp-content/uploads/2023/12/Generative-AI-Nov-2023-Version.pdf.

Becker, B. A., P. Denny, R. Pettit, et al. 2019. "Compiler Error Messages Considered Unhelpful: The Landscape of Text-Based Programming Error Message Research." In *Proceedings of the Working Group Reports on Innovation and Technology in Computer Science Education ITiCSE-WGR 19*, 177–210. New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/3344429.3372508.

Bengtsson, D., and A. Kaliff. 2023. "Assessment Accuracy of a Large Language Model on Programming Assignments." https://urn.kb.se/resolve?urn=urn:nbn:se:kth:diva-331000.

Brocker, A., and U. Schroeder. 2023. "Investigating Feedback Types in JupyterLab for Programming Novices." In *Proceedings of DELFI 2024*, 103–108. Fulda: Gesellschaft für Informatik e.V.

Cipriano, B. P., and P. Alves. 2023. "GPT-3 vs Object Oriented Programming Assignments: An Experience Report." In *Proceedings of the 2023 Conference on Innovation and Technology in Computer Science Education V. 1 ITiCSE 2023*, 61–67. New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/3587102.3588814.

Creswell, A., M. Shanahan, and I. Higgins. 2023. "Selection-Inference: Exploiting Large Language Models for Interpretable Logical Reasoning." In *International Conference on Learning Representations*, Kigali, Rwanda.

Denny, P., V. Kumar, and N. Giacaman. 2023. "Conversing With Copilot: Exploring Prompt Engineering for Solving CS1 Problems Using Natural Language." In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1 SIGCSE 2023*, 1136–1142. New York: ACM. https://doi.org/10.1145/3545945.3569823.

Denny, P., J. Prather, B. A. Becker, et al. 2021. "On Designing Programming Error Messages for Novices: Readability and its Constituent Factors." In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems CHI'21*. New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/3411764.3445696.

Finnie-Ansley, J., P. Denny, B. A. Becker, A. Luxton-Reilly, and J. Prather. 2022. "The Robots Are Coming: Exploring the Implications of OpenAI Codex on Introductory Programming." In *Proc. ACE*, 10–19. New York, NY: Association for Computing Machinery. https://doi.org/10.1145/3511861.3511863.

Hattie, J. 2009. *Visible Learning: A Synthesis of Over 800 Meta-Analyses Relating to Achievement*. London; New York: Routledge. https://doi.org/10.4324/9780203887332.

Hattie, J., and H. Timperley. 2007. "The Power of Feedback." *Review of Educational Research* 77, no. 1: 81–112. https://doi.org/10.3102/003465430298487.

Hellas, A., J. Leinonen, S. Sarsa, C. Koutcheme, L. Kujanpää, and J. Sorva. 2023. "Exploring the Responses of Large Language Models to

Beginner Programmers' Help Requests." In *Proceedings of the 2023 ACM Conference on International Computing Education Research V.1*, 93–105. Chicago IL USA: ACM.

Hill, C. E., S. Knox, B. J. Thompson, E. N. Williams, S. A. Hess, and N. Ladany. 2005. "Consensual Qualitative research: An update." *Journal of Counseling Psychology* 52, no. 2: 196–205.

Hill, C. E., B. J. Thompson, and E. N. Williams. 1997. "A Guide to Conducting Consensual Qualitative Research." *Counseling Psychologist* 25, no. 4: 517–572. https://doi.org/10.1177/0011000097254001.

Ishizue, R., K. Sakamoto, H. Washizaki, and Y. Fukazawa. 2024. "Improved Program Repair Methods Using Refactoring With GPT Models." In *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1 SIGCSE 2024*, 569–575. New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/3626252.3630875.

Jeuring, J., R. Groot, and H. Keuning. 2024. "What Skills Do You Need When Developing Software Using ChatGPT? (Discussion Paper)." In *Proceedings of the 23rd Koli Calling International Conference on Computing Education Research Koli Calling 23*. Association for Computing Machinery: New York, NY, USA. https://doi.org/10.1145/3631802.3631807.

Jeuring, J., H. Keuning, S. Marwan, et al. 2022. "Towards Giving Timely Formative Feedback and Hints to Novice Programmers." In *Proceedings of the 2022 Working Group Reports on Innovation and Technology in Computer Science Education ITiCSE-WGR '22*, 95–115. New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/3571785.3574124.

Joshi, I., R. Budhiraja, H. Dev, et al. 2024. "ChatGPT in the Classroom: An Analysis of Its Strengths and Weaknesses for Solving Undergraduate Computer Science Questions." In *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1 SIGCSE 2024*, 625–631. New York: ACM. https://doi.org/10.1145/3626252.3630803.

Keuning, H., J. Jeuring, and B. Heeren. 2018. "A Systematic Literature Review of Automated Feedback Generation for Programming Exercises." *ACM Transactions on Computing Education* 19, no. 1: 1–43. https://doi.org/10.1145/3231711.

Kiesler, N. 2022a. *Mental Models of Recursion: A Secondary Analysis of Novice Learners' Steps and Errors in Java Exercises*, 226–240. College Station, TX: PPIG.

Kiesler, N. 2022b. "Dataset: Recursive problem solving in the online learning environment CodingBat by computer science students." https://doi.org/10.21249/DZHW:studentsteps:1.0.0 datenerhebung: 2017. Version: 1.0.0. Datenpaketzugangsweg: Download-SUF. Hannover: FDZ-DZHW. Datenkuratierung: İkiz-Akıncı, Dilek. Online.

Kiesler, N. 2022c. "Daten- und Methodenbericht Rekursive Problemlösung in der Online Lernumgebung CodingBat Durch Informatik-Studierende." https://metadata.fdz.dzhw.eu/public/files/data-packages/stu-studentsteps$/attachments/studentsteps_Data_Methods_Report_de.pdf.

Kiesler, N. 2023. "Investigating the Use and Effects of Feedback in CodingBat Exercises: An Exploratory Thinking Aloud Study." In *2023 Future of Educational Innovation-Workshop Series Data in Action*, 1–12. Monterrey, Mexico: IEEE. https://doi.org/10.1109/IEEECONF56852.2023.10104622.

Kiesler, N. 2024. *Modeling Programming Competency: A Qualitative Analysis*. Cham: Springer.

Kiesler, N., D. Lohr, and H. Keuning. 2023. "Exploring the Potential of Large Language Models to Generate Formative Programming Feedback." In *2023 IEEE Frontiers in Education Conference (FIE)*, 1–5. College Station, TX: IEEE. https://doi.org/10.1109/FIE58773.2023.10343457.

Kiesler, N., and D. Schiffner. 2023. "Large Language Models in Introductory Programming Education: ChatGPT's Performance and Implications for Assessments." https://doi.org/10.48550/arXiv.2308.08572.In: CoRR abs/2308.08572. arXiv: 2308.08572.

Kimmel, B., A. L. Geisert, L. Yaro, et al. 2024. "Enhancing Programming Error Messages in Real Time With Generative AI." In *Extended Abstracts of the 2024 CHI Conference on Human Factors in Computing Systems CHI EA'24*. New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/3613905.3647967.

Koutcheme, C., N. Dainese, S. Sarsa, A. Hellas, J. Leinonen, and P. Denny. 2024. "Open Source Language Models Can Provide Feedback: Evaluating LLMs' Ability to Help Students Using GPT-4-As-A-Judge." In *Proceedings of the 2024 on Innovation and Technology in Computer Science Education V. 1 ITiCSE 2024*, 52–58. New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/3649217.3653612.

Kyrilov, A., and D. C. Noelle. 2016. "Do Students Need Detailed Feedback on Programming Exercises and can Automated Assessment Systems Provide It?" *Journal of Computing Sciences in Colleges* 31, no. 4: 115–121.

Lehtinen, T. 2022. "FITech dataset." https://github.com/Programming-Steps-Working-Group-2022/public-datasets/tree/main/FITech.

Leinonen, J., P. Denny, S. MacNeil, et al. 2023. "Comparing Code Explanations Created by Students and Large Language Models." In *Proc. of the 2023 on Innovation and Technology in Computer Science Education (ITiCSE 2024)*, 124–130. Turku, Finland: ACM. https://doi.org/10.1145/3587102.3588785.

Leinonen, J., A. Hellas, S. Sarsa, et al. 2023. "Using Large Language Models to Enhance Programming Error Messages." In *Proc. SIGCSE TS*, 563–569. Toronto, ON: ACM. https://doi.org/10.1145/3545945.3569770.

Lohr, D., and M. Berges. 2021. "Towards Criteria for Valuable Automatic Feedback in Large Programming Classes." In *Hochschuldidaktik Informatik HDI 2021*, 181–186. Dortmund: Jörg Desel, Simone Opel.

Lohr, D., M. Berges, A. Chugh, and M. Striewe. 2024. "Adaptive Learning Systems in Programming Education: A Prototype for Enhanced Formative Feedback." In *Proceedings of DELFI 2024*, 549–554. Fulda: Gesellschaft für Informatik e.V. https://doi.org/10.18420/delfi2024_57.

Lohr, D., N. Kiesler, H. Keuning, and J. Jeuring. 2024. ""Let Them Try to Figure It Out First" - Reasons Why Experts (Do Not) Provide Feedback to Novice Programmers." In *Proceedings of the 2024 Innovation and Technology in Computer Science Education (ITiCSE 2024)*, vol. 1, 7. Milan, Italy: ACM. https://doi.org/10.1145/3649217.3653530.

Luxton-Reilly, A., A. I. Simon, B. A. Becker, et al. 2018. "Introductory Programming: A Systematic Literature Review." In *Proceedings Companion of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education ITiCSE 2018 Companion*, 55–106. New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/3293881.3295779.

Lyulina, E., A. Birillo, V. Kovalenko, and T. Bryksin. 2021. "TaskTracker-Tool: A Toolkit for Tracking of Code Snapshots and Activity Data During Solution of Programming Tasks." In *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education SIGCSE '21*, 495–501. New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/3408877.3432534.

MacNeil, S., A. Tran, A. Hellas, et al. 2023. "Experiences From Using Code Explanations Generated by Large Language Models in a Web Software Development E-Book." In *Proc. SIGCSE TS*, 931–937. Toronto, ON: ACM. https://doi.org/10.1145/3545945.3569785.

Margulieux, L. E., R. Catrambone, and M. Guzdial. 2016. "Employing Subgoals in Computer Programming Education." *Computer Science Education* 26, no. 1: 44–67. https://doi.org/10.1080/08993408.2016.1144429.

Medeiros, R. P., G. L. Ramalho, and T. P. Falcão. 2018. "A Systematic Literature Review on Teaching and Learning Introductory

Programming in Higher Education." *IEEE Transactions on Education* 62, no. 2: 77–90. https://doi.org/10.1109/TE.2018.2864133.

Messer, M., N. C. C. Brown, M. Kölling, and M. Shi. 2024. "Automated Grading and Feedback Tools for Programming Education: A Systematic Review." *ACM Transactions on Computing Education* 24, no. 1: 1–43. https://doi.org/10.1145/3636515.

Narciss, S. 2006. *Informatives Tutorielles Feedback: Entwicklungs- und Evaluationsprinzipien auf der Basis Instruktionspsychologischer Erkenntnisse*. Münster: Waxmann Verlag.

Narciss, S. 2008. "Feedback Strategies for Interactive Learning Tasks." In *Handbook of Research on Educational Communications and Technology*, vol. 3, 125–144. New York: Taylor and Francis Group.

Nguyen, H., and V. Allan. 2024. "Using GPT-4 to Provide Tiered, Formative Code Feedback." In *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1 SIGCSE 2024*, 958–964. New York: ACM. https://doi.org/10.1145/3626252.3630960.

OpenAI. 2023. "GPT-4 Technical Report. arXiv." https://arxiv.org/abs/2303.08774.

OpenAI. 2024. "OpenAI API Documentation: Prompt Engineering." https://platform.openai.com/docs/guides/prompt-engineering.

Paiva, J. C., J. P. Leal, and A. Figueira. 2022. "Automated Assessment in Computer Science Education: A State-Of-the-Art Review." *ACM Transactions on Computing Education*, vol. 22, 1–40. New York, NY, USA: ACM. https://doi.org/10.1145/3513140.

Pankiewicz, M., and R. Baker. 2023. "Large Language Models (GPT) for Automating Feedback on Programming Assignments." In *Proceedings of the 31st International Conference on Computers in Education Matsue*. Shimane, Japan: APSCE.

Phung, T., V. A. Pădurean, A. Singh, et al. 2024. "Automating Human Tutor-Style Programming Feedback: Leveraging GPT-4 Tutor Model for Hint Generation and GPT-3.5 Student Model for Hint Validation." In *Proceedings of the 14th Learning Analytics and Knowledge Conference LAK*, vol. 24, 12–23. New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/3636555.3636846.

Prather, J., P. Denny, J. Leinonen, et al. 2023. "The Robots Are Here: Navigating the Generative AI Revolution in Computing Education." In *Proceedings of the 2023 Working Group Reports on Innovation and Technology in Computer Science Education ITiCSE-WGR'23*, 108–159. New York: ACM. https://doi.org/10.1145/3623762.3633499.

Prather, J., J. Leinonen, N. Kiesler, et al. 2024. "Beyond the Hype: A Comprehensive Review of Current Trends in Generative AI Research, Teaching Practices, and Tools." In *Proceedings of the 2024 Conference on Innovation and Technology in Computer Science Education – Working Group Reports*, vol. 1, 7. Milan, Italy: ACM. https://doi.org/10.1145/3513140.

Prather, J., B. N. Reeves, P. Denny, et al. 2023. ""It's Weird That it Knows What I Want": Usability and Interactions with Copilot for Novice Programmers." *ACM Transactions on Computer-Human Interaction* 31, no. 1: 1–31. https://doi.org/10.1145/3617367.

Qian, Y., and J. Lehman. 2017. "Students' Misconceptions and Other Difficulties in Introductory Programming: A Literature Review." *ACM Transactions on Computing Education* 18, no. 1: 1–24. https://doi.org/10.1145/3077618.

Roest, L., H. Keuning, and J. Jeuring. 2024. "Next-Step Hint Generation for Introductory Programming Using Large Language Models." In *Proceedings of the 26th Australasian Computing Education Conference ACE'24*, 144–153. New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/3636243.3636259.

Sarsa, S., P. Denny, A. Hellas, and J. Leinonen. 2022. "Automatic Generation Of Programming Exercises and Code Explanations Using Large Language Models." In *Proc. of the 2022 ACM Conference on International Computing Education Research*, vol. 1, 27–43. Lugano, Switzerland: ACM. https://doi.org/10.1145/3501385.3543957.

Savelka, J., A. Agarwal, C. Bogart, and M. Sakr. 2023. "Large Language Models (GPT) Struggle to Answer Multiple-Choice Questions About Code." In *Proceedings of the 15th International Conference on Computer Supported Education Prague*, 47–58. Czech Republic: SCITEPRESS - Science and Technology Publications.

Scholl, A., and N. Kiesler. 2024. "How Novice Programmers Use and Experience ChatGPT when Solving Programming Exercises in an Introductory Course." https://arxiv.org/abs/2407.20792, accepted at 2024 IEEE ASEE Frontiers in Education Conference.

Scholl, A., D. Schiffner, and N. Kiesler. 2024. "Analyzing Chat Protocols of Novice Programmers Solving Introductory Programming Tasks With ChatGPT." In *Proceedings of DELFI*, edited by S. Schulz and N. Kiesler, vol. 2024, 63–79. Fulda, Germany: Gesellschaft für Informatik. https://doi.org/10.18420/delfi2024_05.

Shute, V. J. 2008. "Focus on Formative Feedback." *Review of Educational Research* 78, no. 1: 153–189.

Taylor, A., A. Vassar, J. Renzella, and H. Pearce. 2024. "Dcc –help: Transforming the Role of the Compiler by Generating Context-Aware Error Explanations with Large Language Models." In *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1 SIGCSE 2024*, 1314–1320. New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/3626252.3630822.

Wang, S., J. Mitchell, and C. Piech. 2024. "A Large Scale RCT on Effective Error Messages in CS1." In *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1 SIGCSE 2024*, 1395–1401. New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/3626252.3630764.

Wermelinger, M. 2023. "Using GitHub Copilot to Solve Simple Programming Problems." In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1 SIGCSE 2023*, 172–178. New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/3545945.3569830.

Wu, Y., Z. Li, J. M. Zhang, M. Papadakis, M. Harman, and Y. Liu. 2023. "Large Language Models in Fault Localisation." https://doi.org/10.48550/arXiv.2308.15276.

Xiao, R., X. Hou, and J. Stamper. 2024. "Exploring How Multiple Levels of GPT-Generated Programming Hints Support or Disappoint Novices." In *Extended Abstracts of the 2024 CHI Conference on Human Factors in Computing Systems CHI EA'24*. New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/3613905.3650937.

Zamfirescu-Pereira, J. D., R. Y. Wong, B. Hartmann, and Q. Yang. 2023. "Why Johnny Can't Prompt: How Non-AI Experts Try (and Fail) to Design LLM Prompts." In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems CHI '23*. New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/3544548.3581388.