AI in Programming Education: Automated Feedback Systems for Personalized Learning

YULIYA ALYOSHYNA, University of Twente, The Netherlands

Feedback is an important aspect of a balanced learning program, significantly improving student performance and engagement. In computer science education, especially in bachelor's programs, feedback is essential for learning programming languages and problem-solving skills. This research explores the integration of artificial intelligence (AI) into feedback systems for coding exercises, addressing current challenges in manual feedback, such as timeliness, relevance, and the teacher's workload. The research examines the characteristics of programming exercises, the current methods and technologies used in AI-driven feedback systems, and the potential benefits and limitations of such systems. A semi-structured literature review and experimental analysis were conducted to evaluate the potential of automated feedback mechanisms. Results indicate a preference for concise programming tasks focusing on foundational concepts, with AI-driven feedback systems showing promise in scalability and personalization. However, limitations in feedback quality and context understanding were identified.

Additional Key Words and Phrases: artificial intelligence, intelligent tutoring systems, automated feedback, computer-assisted learning

1 INTRODUCTION

Providing feedback to students is an essential part of any learning curriculum. Feedback guides students in improving their work, understanding their mistakes, supports independence in learning, and improves student's performance [29].

In computer science bachelor's programs, proficiency in programming languages and problem-solving is important for students to become well-rounded computer scientists [10, 31]. They develop the skills through assignments, tutorials, lab sessions, and projects, with immediate feedback being critical in the early stages, especially for coding tasks [3]. Early identification and correction of errors are essential for students to be successful in their future work [29].

However, currently, there are challenges in providing feedback such as student dissatisfaction due to content issues, timeliness, and relevance of feedback comments [7]. Teachers also face increasing pressure and students' unwillingness to use feedback. Despite efforts by institutions, feedback remains poorly understood and implemented [7].

The modern solution to this problem is automated feedback systems. Automated feedback refers to using technology, particularly artificial intelligence (AI) and machine learning (ML), to provide feedback on student work without explicit human intervention. This feedback can be delivered in different forms, such as:

- Multiple-choice question grading
- Code review
- Writing evaluation
- Open-ended question analysis

These methods refer to identifying correct answers, highlighting errors, suggesting improvements, and using natural language processing (NLP) for content analysis.

The main objective of this research is to explore the integration of artificial intelligence (AI) into feedback systems that focus on programming exercises within higher-education settings. This includes analysing the characteristics of programming exercises, analysing current methods and technologies used to incorporate AI into feedback systems, and evaluating the potential benefits and limitations of using AI in automated feedback systems for programming exercises.

2 PROBLEM STATEMENT

There are challenges with automated feedback for programming exercises [4, 11, 12]. The integration of AI into educational courses has shown promising potential to improve traditional feedback practices, particularly in the context of programming exercises. However, despite the advancements in AI technologies, several challenges persist in incorporating AI-driven feedback systems into coursework.

Secondly, current methods and technologies used to integrate AI into feedback systems need to be examined. Understanding the strengths, weaknesses, and applicability of these methods is essential for developing effective feedback systems.

Furthermore, evaluating automated feedback has challenges. While most tools offer some evaluation, the quality and clarity of these methods and results vary greatly [12]. Inconsistent evaluation methods and insufficient detail in descriptions make assessing feedback effectiveness difficult. While AI-driven feedback systems offer the promise of better adaptiveness, personalization, and effectiveness in supporting student learning outcomes, some existing challenges and limitations need to be addressed.

2.1 Research question

Thus, following from the problem statement, we can formulate our research question as follows:

RQ How can artificial intelligence be used to provide immediate feedback for programming exercises?

To answer this research question, I will further split it into the following sub-questions:

SRQ1 What characteristics do programming exercises have?
SRQ2 What are the current methods and technologies used to incorporate artificial intelligence for providing feedback for programming exercises?
SRQ3 What are the potential benefits and limitations of using artificial intelligence in feedback systems for programming exercises?

TScIT 41, July 7, 2024, Enschede, The Netherlands

A small experiment involving feedback generation using Large Language Models (LLMs) and student participants will be conducted to evaluate and understand student perceptions of AI-generated feedback. This experiment aims to determine whether students find the feedback useful and if they are interested in the integration of such systems into computer science bachelor's programs.

3 SRQ1 RESULTS — CHARACTERISTICS OF PROGRAMMING EXERCISES

The first sub-research question (SRQ1) focuses on understanding the characteristics of programming exercises, which are important for developing automated feedback systems. When training a machine learning model or developing rule-based logic for automated feedback, these characteristics are the primary focus for the analysis of a student's solution.

3.1 Approach

To gain a better understanding of how Bachelor's Computer Science programming exercises are formulated, a framework was developed to identify and categorize the most common programming task requirements. This framework is summarized in Table 1. Using this framework, Java exercises from the Software Systems (Module 2) first-year course of the Bachelor of Technical Computer Science (TCS) program were manually labelled and categorized. The results of this analysis are discussed in Section 3.2.

Further, an experimental approach was used, applying data science and natural language processing techniques to identify the most common words and phrases in programming task descriptions and their typical instruction lengths. The experiment was performed on two different datasets:

- **Module 2 TCS**:
  The dataset was created by selecting diverse programming exercises from the Bachelor of Technical Computer Science Software Systems (Module 2) course guide, covering a range of topics such as arithmetic operations, recursion, object-oriented programming, sorting algorithms, and basic security. Module 2 focuses on teaching students Object Oriented Programming (OOP) through Java exercises.
- **HuggingFace dataset**:
  A larger dataset sahil2801/CodeAlpaca-20 of diverse programming exercises available online from Hugging Face datasets. The dataset contained programming tasks for various programming languages, topics, and solutions.

The results of the experiment for both datasets are discussed in Section 3.3.

3.2 Framework Classification Results

During the classification process, it was observed that the exercises were incremental in difficulty, starting with basic tasks and gradually becoming more complex. The emphasis was on understanding the logic of the exercises rather than just the syntax. Students were encouraged to test their solutions to ensure correctness, although there was often little guidance on code documentation or comments, highlighting a potential area for feedback improvement.

Many exercises required simple input/output verification for correctness, making them suitable for expert-driven feedback approaches, which rely on predefined rules and criteria set by human experts, such as testing for correct output via the use of test cases (Discussed in Section 4.3).

The exercises began with fundamental concepts, such as taking user input and printing it out, performing mathematical operations, reading/writing from/to a file, and learning about data structures, thus providing students with a solid foundation before introducing Object-Oriented Programming (OOP). Subsequently, the exercises primarily focused on OOP, along with some security and networking concepts.

Diverse problem-solving strategies were sometimes possible, emphasizing the need for correct base logic regardless of the implementation. However, in most cases, the implementation would only differ slightly (e.g., different variable names or using a `'while'` loop instead of a `'for'` loop). Some exercises allowed for multiple implementation variants and different solution strategies, such as in network programming concepts, where the main objective was to implement a specific programming pattern.

For effective instruction, exercises required clear and concise descriptions, learning objectives, hints for solutions, tool suggestions, descriptive steps or resources, test cases, and indications of expected results. Solutions needed to address the correct problem as required per the task description, be syntactically correct, use appropriate tools, pass test cases, meet input/output requirements, and be in text format.

3.3 Experimental Results

3.3.1 Module 2 TCS. Module 2 TCS exercises focus exclusively on the Java programming language. The majority of task descriptions are concise, typically ranging between 400 and 600 characters. There are fewer problems with longer instructions, indicating a preference for shorter problem statements.

Analysis of tokens and common phrases provided limited insights. Common phrases often reflect the instructional and action-oriented nature of the exercises, indicating the tasks students are asked to perform. The most frequently occurring phrases were `'write'`, `'program'`, `'user'`, `'write program'`, `'run'`, and `'example'`. This suggests a hands-on approach to learning programming through practical examples and user interaction.

In contrast, common tokens reveal the key elements and concepts central to the tasks. The most common tokens identified were `'number'`, `'user'`, `'method'`, `'program'`, and `'class'`, highlighting frequently addressed concepts in the exercises.

While the presence of instructional phrases is expected, the frequent occurrence of certain tokens does not provide clear conclusions about the specific concepts covered in the exercises. Overall, both tokens and phrases suggest a focus on fundamental programming concepts such as handling numbers, user input, and method implementation, which are essential for building a solid foundation in programming.

3.3.2 HuggingFace Dataset. A similar experiment was conducted using an online found dataset, revealing that the majority of exercises solutions are expected to be written in C programming

Table 1. Programming exercises classification framework

Criteria	Parameters description
Task type	Nature of the task such as reading, writing, debugging, refactoring
Input/output requirements	Expected input-output format: **Simple I/O:** Basic user input (e.g., numbers, strings) and console output. **File Processing:** Reading/writing data from/to files. **Network Programming:** Interaction with network protocols or APIs.
Number of solutions	Possible ways to complete the task: **Single solution:** One correct output per input. **Multiple Variants:** Different implementations achieve the same outcome. **Different Strategies:** Various problem-solving techniques. **Open-ended:** Loosely defined problems, encouraging creativity.

language. This emphasizes a focus on foundational programming concepts, as C is commonly used to teach basic programming and system-level concepts. Python and Java follow in popularity, reflecting their widespread use in both educational settings and industry. Python's simplicity and readability make it ideal for beginners, while Java is extensively used for teaching object-oriented programming.

The distribution of instruction lengths indicates that most exercises feature short instructions. This suggests that the exercises are designed to be brief, potentially allowing students to quickly understand the problem requirements.

Common instructions include tasks such as generating random numbers, creating loops, and evaluating expressions, highlighting a focus on basic programming constructs and algorithmic thinking. Other frequent phrases, such as `'write function'`, `'given string'`, and `'write code'`, primarily reflect the instructional language used to direct students to solve the exercise.

3.4 Comparison

Analysing the Module 2 TCS, HuggingFace datasets and framework classification analysis revealed several similarities and insights into the characteristics of programming exercises. Firstly, both approaches showed a preference for shorter problem statements. Secondly, a hands-on approach to learning programming through practical examples and user interaction was suggested in both approaches. Furthermore, both approaches have shown some similarities in a focus on basic programming concepts.

Overall, the findings from both approaches emphasize the importance of concise, practical, and focused on programming fundamentals exercises in developing automated feedback systems.

4 SRQ2 RESULTS — CURRENT METHODS AND TECHNOLOGIES IN AUTOMATED FEEDBACK FOR PROGRAMMING EXERCISES

The second sub-research question (SRQ2) investigates the current methods and tools that incorporate artificial intelligence and other technologies to provide automated feedback for programming exercises.

4.1 Approach

A semi-structured literature review was conducted to answer the research question. To locate papers on automated feedback, AI, and programming exercises, search engines such as Google Scholar, Scopus, IEEE Xplore, and ACM Digital Library were used. The following search query was developed:

```
( TITLE-ABS-KEY ( ( automated AND feedback )
AND education
AND ( "computer science" OR programming OR coding
OR "coding exercises" )
OR "intelligent tutoring systems" )
AND KEY ( "machine learning" OR "intelligent systems"
OR "intelligent tutoring systems"
OR "automated feedback"
OR "adaptive learning systems" ) ) AND PUBYEAR > 2019
AND PUBYEAR < 2025 AND ( LIMIT-TO ( SUBJAREA, "comp" ) )
```

This query aimed to find publications from 2020 to 2024 in computer science, focusing on automated feedback in programming education. The reason for selecting the 4-year range was to ensure the inclusion of the most recent and relevant studies. Additionally, some papers within this timeframe already include literature reviews of earlier works, providing a helpful overview of previous research. The initial query yielded 74 papers, which were refined by reading titles and abstracts, resulting in 35 papers for detailed analysis. These papers were analysed for techniques and technologies in automated feedback systems for computer science programming exercises.

Figure 1 summarizes the methods and technologies identified in the literature review, highlighting three main approaches: data-driven, expert-driven, and mixed. Each approach is described in the following sections, data-driven in Section 4.2, expert-driven in Section 4.3 and mixed approach in Section 4.4.

4.2 Data-Driven Approach

Data-driven approaches derive feedback rules from student data using algorithms and machine learning. Machine learning models in intelligent tutoring systems typically require large amounts of historical data to provide feedback.

According to Deeva et al. (2021) [4], 32% of reviewed studies by them focused solely on data-driven models. According to multiple studies [4, 5, 8, 12, 14, 15, 17, 18, 20, 21, 24, 26], machine learning approaches are varied and depend on the desired outcomes.

In this section, different methods are described according to whether they are focused on extracting features, analysing data

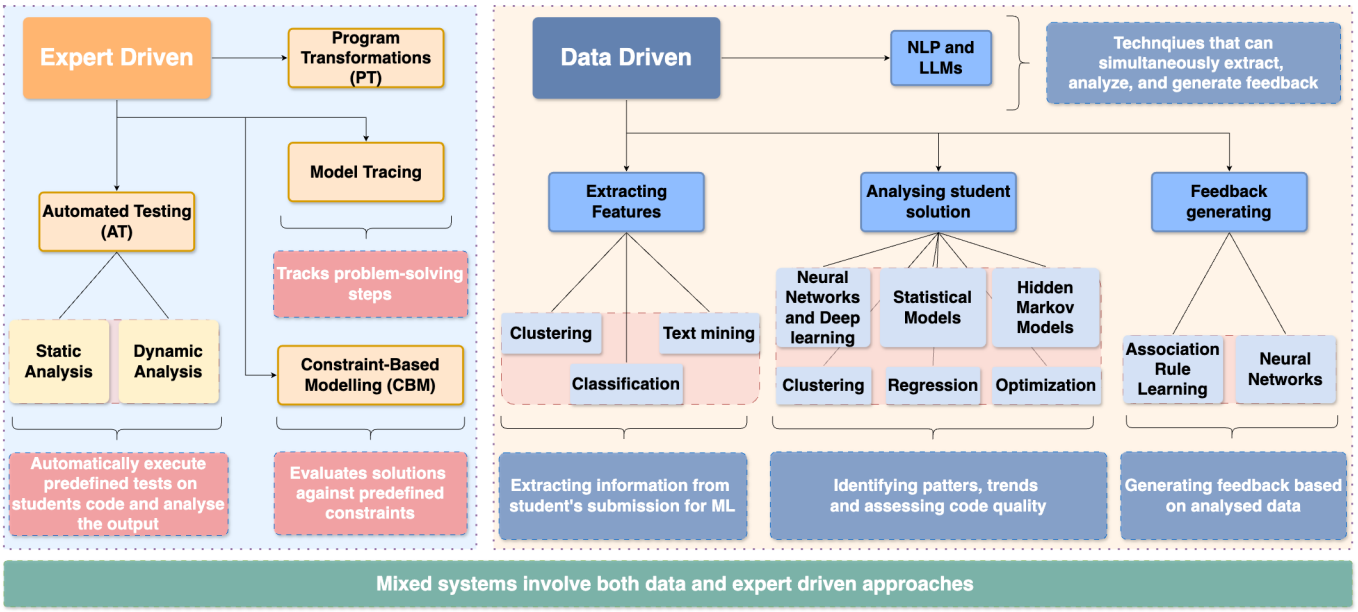

Fig. 1. Methods of Automating Feedback in Programming Education

or generating feedback. Natural language processing and large language models are discussed separately as they combine all three: extraction, analysis, and feedback generation.

4.2.1 Extracting Information. To provide feedback, relevant information must be extracted from student submissions. Key methods include:

- **Classification**: Techniques like support vector machines, naive Bayes, random forests, nearest-neighbour algorithms, logistic regression, and decision trees. These methods classify new data based on labelled input and identify relevant code patterns across students [4, 17].
- **Text mining**: Extracts useful information from text, such as context and intent from code comments to provide relevant feedback.
- **Clustering**: Groups similar data points to identify common patterns and misconceptions, such as clustering programs by functionality to understand student behaviour [19].

The extracted data can be used as features for further machine learning and analysis, including comments, intent, and programming style.

4.2.2 Analysing. Analysis is critical for automated feedback systems, enabling the understanding, evaluation, and improvement of student submissions. It helps identify patterns, trends and assess code quality for later feedback generation. Key methods include:

- **Neural Networks and Deep Learning**: Techniques such as convolutional neural networks (CNN), recurrent neural networks (RNN), Long Short-Term Memory (LSTM), and encoder/decoder models. Dense neural networks can be used to predict how an incorrect solution can be repaired, and CNN can be used to evaluate assignment quality based on underlying semantic structure [22]. Neural networks can also be used to predict the outcome of the program [26].
- **Statistical-based models**: These models can be used in identifying trends and predicting student performance based on historical data [4].
- **Optimization**: Algorithms like gradient descent and Monte Carlo methods can optimize the performance and accuracy of feedback systems [4].
- **Regression**: Regression models can predict a continuous outcome variable based on one or more predictor variables. For example, they can be used in analysing relationships between different variables in student performance data to predict scores [4, 17, 22, 26].
- **Hidden Markov Models**: Modelling sequences of data points. These can be used in analysing sequences of student actions to identify common problem-solving patterns [4, 12, 19].
- **Sequence Mining**: Identifying sequences and patterns within data, for example, to detect common sequences in code submissions that lead to errors or successes [4].

Analysed student solution produces a basis for generated feedback whether it is a suggestion on improving code quality, a suggestion on how to fix a mistake or a hint for further solutions to the task.

4.2.3 Generating. The final step of automated feedback would be the generation of feedback. Feedback generation involves methods similar to analysis:

- **Neural networks**: Based on the analysis, neural networks can generate personalized hints and suggestions. This involves providing specific guidance on how to correct errors or improve code quality [2, 12, 22].

- **Association rule learning**: Association rule learning could generate feedback by discovering associations between different aspects of student submissions and learning outcomes. However, typically, expert insight would be needed to develop the association rules [4].

Typically, the generated feedback is in the form of text; however, some feedback tools, such as the one mentioned by Messer et al. (2024) and developed by Edmison and Edwards, combine spectrum-based fault localization with visualization. This tool provides feedback on maintainability and correctness by using unit testing and visualizing the spectrum analysis as a heatmap, where more suspicious code produces a higher score [22].

4.2.4 Natural language processing (NLP) and Large Language Models (LLMs). NLP techniques and LLMs can simultaneously extract, analyse, and generate feedback, making them useful tools in automated feedback systems.

NLP techniques have gained popularity in automated research in computer science [24]. In automated feedback systems, NLP can engage in dialogue with students for planning and program design [12]. It can assess source code [20] and code comments [16], extracting useful information to generate feedback [33]. For instance, Kochmar et al. (2021) used NLP to generate hints and a random forest classifier to determine their appropriateness, resulting in personalized feedback [15].

LLMs, like OpenAI's GPT series, understand and generate human language text. Messer et al. (2024) noted the rise of LLMs in computer science education [24]. LLMs, such as ChatGPT, generate context-aware and personalized feedback in human-understandable language. They can learn from feedback loops without retraining, which makes them very convenient. Pankiewicz and Baker (2024) used GPT-3.5 to generate personalized hints for programming exercises [27]. Roest et al. (2024) explored using LLMs to generate next-step hints for students [28].

While NLP and LLMs have powerful capabilities in extracting, analysing, and generating feedback, they also have limitations, those limitations are discussed in section 5.3.1.

4.3 Expert-Driven Approach

Expert-driven approaches rely mainly on expert knowledge to generate feedback rules. For example, Deeva et al. (2021) points out that "expert knowledge will be potentially transformed to a set of rules (a feedback generation model), such as 'if a student's answer is 5/10 instead of 1/2, the system should suggest repeating section 17 in the textbook with an explanation about reducing equivalent fractions to the lowest terms.'"

Typically, feedback is predefined for each issue found in a student's code. The main task is to verify whether the student's solution meets specific criteria, using the following approaches:

4.3.1 Automated Testing (AT) and Static/Dynamic Approaches. Automated Testing (AT) uses software tools to automatically execute predefined tests on students' code, assessing correctness, efficiency, and coding standards. As Keuning et al. (2018) noted, "AT is often implemented by running a program and comparing its output to the expected output" [12].

Integrating AT with static and dynamic approaches is common for automating programming exercise assessments, focusing on correctness and feedback [21, 24].

- **Static Analysis**: Examines source code without execution to detect errors, maintainability issues, security vulnerabilities, and coding standards adherence [12, 23]. It often compares student submissions with reference solutions to identify discrepancies [23]. Tools include PyLint, cpplint, CheckStyle, FindBugs, and PMD [23].
- **Dynamic Analysis**: Executes code to analyse runtime behaviour, testing performance, memory usage, and correct operation under various conditions [12].

Combining AT with static or dynamic analysis offers more detailed feedback, as AT alone only provides test case results without specific issue feedback [12].

4.3.2 Intelligent Tutoring Systems (ITS) with Model Tracing and Constraint-Based Modelling. Another technique is model tracing and constraint-based modelling, those techniques can identify where the student is in his programming task and generate hints on the next steps.

- **Model Tracing**: Tracks and analyses students' solution's steps against predefined production and common erroneous rules. It helps diagnose specific misconceptions and provides targeted feedback [12].
- **Constraint-Based Modelling (CBM)**: Evaluates the final solution based on whether it meets predefined constraints, useful in domains with multiple acceptable solution paths [12].

4.3.3 Program Transformations (PT). Program transformations involve modifying source code to improve performance, readability, correctness, or structure without changing functionality. This method helps students learn best practices and coding standards. When combined with static analysis, program transformations can align student programs with example programs [13]. Tools like FindBugs use static analysis to identify potential bugs by translating Java programs into bytecode [13].

Additionally, repair generators can apply transformation patterns to faulty programs to create and validate potential fixes using the provided test suite, subsequently offering solutions or suggestions to students [36].

4.4 Mixed Approach

Mixed approaches involve both data and expert-driven approaches, where a mixed feedback system defines its rules by integrating expert knowledge with insights gained from analysing student data [4].

Zhan et al. (2022) discuss a hybrid feedback generation system that combines a formal rule-based method with a data-based multi-label classification (MLC) technique. The system predicts user intent and provides examples of successful solutions, demonstrating its effectiveness in assisting users with troubleshooting and task completion [35].

An example of using both data-driven and expert-driven approaches is mentioned by Messer et al. (2024): "To grade correctness,

Dong et al. [18] implemented two ML AATs into an online judge. The first model was trained using historical training data to predict what causes failed test results. The second model implemented was a knowledge-tracing model used to predict the probability of a student passing a new problem based on previous knowledge of a programming concept" [23].

4.5 Applicability and Use of Automated Feedback Approaches

As mentioned, data-driven methods use previous data and machine learning algorithms to generate feedback. This approach applies in environments with a large amount of historical data available. For example, it can predict student performance [18, 23] identify common errors [17], and provide personalized recommendations based on provided data [34]. Data-driven approaches are suitable for scenarios where personalized feedback needs to handle a wide range of student interactions, thus making it more generalizable across different scenarios [1, 15]. Data-driven methods are scalable, making them well-suited for delivering feedback to a large number of students. [8].

Expert-driven methods rely on predefined rules and expert knowledge to generate feedback. Unlike data-driven approaches, expert-driven systems don't require large datasets for training. This makes them suitable for situations with limited student data or when the focus is on specific, well-defined concepts [12]. Expert-driven approaches generally do not provide diverse feedback and are effective in tools that primarily focus on identifying mistakes made by learners [12]. Thus, making them applicable in scenarios where consistency of feedback is important and/or identification of mistakes is a priority.

Mixed approaches use both data and expert-driven approaches, making them suitable in scenarios where both a large amount of data and expert knowledge is available.

5 SRQ3 RESULTS — BENEFITS AND LIMITATIONS OF AI IN AUTOMATED FEEDBACK SYSTEMS

The third sub-research question (SRQ3) focuses on evaluating the benefits and limitations of AI in automated feedback systems in programming education. Understanding these aspects is crucial for comprehending the overall effectiveness and potential drawbacks of integrating such technologies in educational contexts.

5.1 Approach

Similarly to the approach in SRQ2, a semi-structured literature review was conducted. Relevant studies found from SRQ2 were analysed to identify benefits and limitations of AI in automated feedback systems.

5.2 Benefits

Automated feedback systems provide immediate feedback on programming assignments, which is crucial for learning. Immediate feedback helps students correct their mistakes promptly and reinforces learning concepts effectively [23]. Expert-driven automated systems ensure consistency and objectivity in grading and feedback due to defined rules for providing feedback. Unlike human graders, these systems do not suffer from fatigue or bias, leading to fairer assessments [12], however, that might not always be the case with data-driven approaches if the data supplied is already biased.

AI-driven feedback systems can handle large volumes of student submissions efficiently, making them particularly useful in Massive Open Online Courses (MOOCs) and large university classes where manual grading would be impractical, thus saving resources. This scalability helps in providing consistent feedback to all students regardless of class size [6].

By leveraging data-driven techniques and machine learning, automated feedback systems can provide personalized feedback tailored to individual student needs, improving learning outcomes [4]. Some advanced automated feedback AI systems integrate motivational and meta-cognitive feedback, encouraging students to develop better learning strategies and fostering a more positive learning environment [6].

5.3 Limitations

While automated feedback systems can provide quick feedback, the quality and depth of the feedback may not match that of experienced human instructors. AI automated feedback can sometimes be generic or miss nuanced errors that require human judgment [22].

The development and maintenance of AI automated feedback systems require significant technical expertise and resources due to the rapidly evolving field of AI. Implementing these systems can be costly and time-consuming [25].

Expert-driven automated feedback often relies heavily on predefined test cases. This can lead to situations where students' code passes all test cases, but still contains underlying issues not captured by the tests [23]. AI automated systems may struggle to understand the broader context of a student's work or the pedagogical goals behind an assignment, especially if it's not mentioned in the assignment. This limitation can result in feedback that is not fully aligned with the learning objectives [6]. In cases where the system misinterprets the code or the intended solution, it can provide incorrect or misleading feedback, confusing students and hindering their learning process [23].

5.3.1 NLP and LLMs limitations. One significant limitation is the need for large, high-quality datasets for training, which may not always be available in educational contexts [32]. Additionally, LLMs can sometimes produce contextually inappropriate or incorrect feedback if the input data is ambiguous or if the model has not been adequately fine-tuned [9, 30]. Furthermore, these models often require substantial computational resources, which can be a barrier to their widespread adoption in educational settings. Lastly, while LLMs can generate human-like text, they may lack the deep domain-specific knowledge and pedagogical insights that expert-driven approaches provide, potentially limiting their effectiveness in certain educational scenarios [9].

6 EXPERIMENT EVALUATING LLM-GENERATED FEEDBACK ON PROGRAMMING EXERCISES

The primary goal of this experiment is to evaluate the usefulness and potential of Large Language Models (LLMs) in automating feedback

for programming exercises by comparing different LLMs based on student feedback.

6.1 Experimental design

To evaluate the usefulness and potential of LLMs in automating feedback for programming exercises, the experiment was structured as follows:

6.1.1 Selection of Programming Exercises. 10 different programming exercises from Bachelor's of Technical Computer Science Software Systems (Module 2) course were selected. The exercises covered a range of topics from basic arithmetic operations and data structures to more advanced concepts like recursion, object-oriented programming, sorting algorithms, and data security.

6.1.2 Selected LLMs. Four well-known and accessible LLMs were chosen for evaluation:

- **OpenAI GPT-3.5**: Advanced natural language understanding and generation capabilities, trained on large amount of data. Well-suited for generating detailed and context-aware response.
- **Copilot**: An AI-powered code completion tool developed by GitHub and OpenAI, specifically trained on a vast amount of programming-related data.
- **Google Gemini**: A language model developed by Google, known for its robust performance in understanding and generating human-like text.
- **Hugging Face model**: CohereForAI/c4ai-command-r-plus is a versatile model from Hugging Face's repository, designed to handle language tasks.

6.1.3 Procedure. The selected programming exercises were collected from five first-year Computer Science students who are familiar with programming basics. The exercise instructions and their solution were provided to the selected LLMs using the following prompt:

```
Generate a short concise feedback on the solution.
Problem: [Problem description inserted here]
Solution: [Solution code inserted here]
```

After receiving the feedback generated by the LLMs, students were asked to fill out an evaluation form.

6.2 Evaluation form

The evaluation form for assessing the effectiveness and usefulness of feedback provided by LLMs for programming exercises was based upon some principles from the Unified Theory of Acceptance and Use of Technology (UTAUT) and the Technology Acceptance Model (TAM).

- **Performance Expectancy/Perceived Usefulness**: Measured through quantitative ratings on the usefulness, clarity, accuracy, and helpfulness of the feedback, reflecting how well the feedback helps students understand and improve their coding skills.
- **Perceived Ease of Use**: Assessed through qualitative feedback and quantitative clarity evaluations, capturing how easy it is for students to understand and apply the feedback.
- **Social Influence and Behavioural Intention to Use**: Defined through additional comments on whether students believe LLM automated feedback systems should be integrated into their coursework and their reasons for this belief.

6.3 Results

The collected evaluation forms were analysed to compare student's opinions on the different LLMs provided feedback. The analysis focused on identifying which LLM provided the most effective feedback and understanding the reasons behind the students' preferences.

6.3.1 OpenAI GPT-3.5. Student evaluations of GPT-3.5 feedback revealed mixed results. While students appreciated the structured and detailed explanations, significant issues with accuracy and relevance were noted, resulting in low usefulness ratings. Clarity and helpfulness ratings varied, with feedback often lacking in value. Students criticized circular reasoning and irrelevant text, recommending improvements in focus, conciseness, accuracy, and better code detail checking.

6.3.2 Copilot. Copilot feedback received generally neutral to positive evaluations. Students valued its clarity, structured approach, and focus on small mistakes, with high ratings for accuracy and helpfulness. However, issues included occasional misunderstandings and unimplemented code revisions. Students suggested more detailed explanations and ensuring actual changes in revised code.

6.3.3 Google Gemini. Evaluations of Google Gemini feedback were mixed. Some students appreciated the clear, structured breakdown into categories, while others were dissatisfied with its usefulness and accuracy. Positive aspects included clear categorization and concise comments, but significant issues involved repetitive feedback and suggestions for non-existent methods.

6.3.4 CohereForAI/c4ai-command-r-plus model. Evaluations of Hugging Face model were mostly neutral to negative. Students liked the focused responses, but noted problems with accuracy and structure, leading to mixed usefulness ratings. Clarity ratings were also mixed, with feedback often lacking informativeness and organization. Helpfulness ratings indicated the feedback frequently failed to be helpful and guide students in the right direction. Students suggested improvements in structuring feedback and providing more detailed guidance.

6.3.5 Comparative Analysis. Copilot received the highest satisfaction ranking with 4/5 students rating it as the most useful, followed by Google Gemini with a score of 1/5. Hugging Chat was rated by all students as the least useful. Copilot was preferred for its structured, accurate, and clear feedback, while Google Gemini was valued by one student for thorough explanations and mostly accurate feedback. Hugging Chat was rated the least useful due to its lack of useful suggestions and general comments.

Opinions on integrating LLM-generated feedback into coursework were mixed. Some students supported it for helping understand and correct mistakes efficiently, while others preferred human review for accuracy and relevance.

Additional comments included the suggestion to develop educational specific AI tools with official references to avoid misleading information and plagiarism. There was also a call for human supervision alongside AI feedback to enhance its reliability and effectiveness.

6.4 Evaluation

The experiment showed that students valued clear, concise, and actionable feedback, they appreciated explanations that directly addressed their code's problems and offered solutions. Feedback containing irrelevant information, too general or suggesting incorrect solutions was considered unhelpful.

Overall, LLMs demonstrate promising capabilities in generating relevant feedback that is useful for students, particularly when fine-tuned on programming data, as observed in this experiment with Copilot. There's a need for improvement in accuracy, focus, and providing actionable feedback. Human review is likely still necessary to ensure quality feedback.

The speed of feedback generation through LLMs is significantly better than manual methods, offering a distinct advantage. However, practical limitations such as character and message constraints constrained the dataset selection to shorter exercises, affecting the level of analysis. Moreover, limitations on the amount of feedback that could be generated at a time due to message restrictions indicate the importance of resource optimization for implementing such tools.

Finally, it is important to note that the results cannot be generalized to a larger population or for all programming exercises, but they do provide insight into how AI is capable of generating immediate feedback for programming exercises.

7 FUTURE WORK AND LIMITATIONS

Regarding the performed experiment, future research should aim to collect a larger and more diverse dataset of programming exercises to ensure findings are generalizable across different educational contexts. Expanding the dataset to include a wider range of programming tasks, different programming languages, various contexts, and diverse student demographics will provide more insights into the effectiveness of LLM-generated feedback. Additionally, developing more programming-specific language models that better understand and address specific coding issues is crucial. This could involve integrating advanced natural language processing techniques and domain-specific knowledge bases to improve feedback relevance. Exploring mixed models that combine the strengths of data-driven and expert-driven approaches could also provide more insights into the applicability of mixed approaches models.

Currently, there is a gap in research on how generated feedback could be personalized. With the potential of data-driven approaches, personalization should be investigated to see how feedback systems can change responses based on individual student learning styles, progress, previous submissions or other variables. Personalized feedback tools can better address the unique needs of each student, potentially improving learning outcomes. Research should also explore the use of student feedback and performance data to continually refine and customize the machine learning models, making them more adaptable and effective over time.

Ethical considerations are a significant part of AI research. Due to time constraints, this study did not address ethics, but future research should investigate methods to ensure ethical use and address potential biases in AI-generated feedback.

This research was also limited by a relatively small number of participants in the experiment, variability in student perceptions, a narrow focus on immediate reactions and technical limitations of LLMs. Addressing these limitations in future work will allow the generalization of findings and more revealing insights about the use of automated feedback for programming exercises.

8 CONCLUSION

This research explored the integration of artificial intelligence into feedback systems for programming exercises in higher education. By analysing the characteristics of programming exercises and evaluating the current methods and technologies used in AI-driven feedback systems, the study highlighted both the potential benefits and limitations of these systems.

Characteristic analysis of programming exercises showed a preference for concise problem statements with a focus on basic programming concepts.

A short literature review showed that data-driven approaches offer scalable feedback but face challenges with data availability, deep domain-specific knowledge, and accuracy of feedback. Expert-driven approaches provide consistent and reliable feedback but can lack the depth of feedback or miss underlying issues with the student solution.

The evaluation of various LLMs, including OpenAI GPT-3.5, Copilot, Google Gemini, and Hugging Chat (CohereForAI/c4ai-command-r-plus), revealed mixed student perceptions. While Copilot and Google Gemini were generally well-received for their structured and accurate feedback, Hugging Chat was criticized for its lack of useful suggestions and the presence of general comments.

Future work should focus on expanding datasets for analysing programming exercises characteristics, developing more specialized language models for programming, and ensuring ethical considerations of AI in feedback systems. Addressing current limitations, such as dataset size, variability in student perceptions, and technical constraints, will further improve the evaluation of the effectiveness of LLM-generated feedback in programming education. Integrating these systems into educational platforms, with a focus on personalization and ethical use, might improve student learning outcomes and engagement.

REFERENCES

[1] Muhammad Afzaal, Jalal Nouri, Aayesha Zia, Panagiotis Papapetrou, Uno Fors, Yongchao Wu, Xiu Li, and Rebecka Weegar. 2021. Explainable AI for data-driven feedback and intelligent action recommendations to support students self-regulation. *Frontiers in Artificial Intelligence* 4 (2021), 723447.

[2] U.Z. Ahmed, N. Srivastava, R. Sindhgatta, and A. Karkare. 2020. Characterizing the pedagogical benefits of adaptive feedback for compilation errors by novice programmers. *Proceedings - International Conference on Software Engineering* (2020), 139–150. https://doi.org/10.1145/3377814.3381703 cited By 18.

[3] Albert T Corbett and John R Anderson. 2001. Locus of feedback control in computer-based tutoring: Impact on learning rate, achievement and attitudes. In *Proceedings of the SIGCHI conference on Human factors in computing systems*. 245–252.

[4] Galina Deeva, Daria Bogdanova, Estefanía Serral, Monique Snoeck, and Jochen De Weerdt. 2021. A review of automated feedback systems for learners: Classification framework, challenges and opportunities. *Computers & Education* 162 (2021), 104094.

[5] E.M. Dillon, C. Carpenter, J. Cook, T.D. Wills, and H.S. Narman. 2022. A Machine Learning-Based Automatic Feedback System to Teach Cybersecurity Principles to K-12 and College Students. *2022 IEEE Global Humanitarian Technology Conference, GHTC 2022* (2022), 219–225. https://doi.org/10.1109/GHTC55712.2022.9910998 cited By 1.

[6] Hagit Gabbay and Anat Cohen. 2023. Unfolding Learners' Response to Different Versions of Automated Feedback in a MOOC for Programming–A Sequence Analysis Approach. *International Educational Data Mining Society* (2023).

[7] Michael Henderson, Tracii Ryan, and Michael Phillips. 2019. The challenges of feedback in higher education. *Assessment & Evaluation in Higher Education* (2019).

[8] M. Hooda, C. Rana, O. Dahiya, A. Rizwan, and M.S. Hossain. 2022. Artificial Intelligence for Assessment and Feedback to Enhance Student Success in Higher Education. *Mathematical Problems in Engineering* 2022 (2022). https://doi.org/10.1155/2022/5215722 cited By 51.

[9] Andrew Katz, Umair Shakir, and Ben Chambers. 2023. The Utility of Large Language Models and Generative AI for Education Research. *arXiv preprint arXiv:2305.18125* (2023).

[10] Hieke Keuning, Bastiaan Heeren, and Johan Jeuring. 2017. Code quality issues in student programs. In *Proceedings of the 2017 ACM Conference on Innovation and Technology in Computer Science Education*. 110–115.

[11] Hieke Keuning, Johan Jeuring, and Bastiaan Heeren. 2016. Towards a systematic review of automated feedback generation for programming exercises. In *Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education*. 41–46.

[12] Hieke Keuning, Johan Jeuring, and Bastiaan Heeren. 2018. A systematic literature review of automated feedback generation for programming exercises. *ACM Transactions on Computing Education (TOCE)* 19, 1 (2018), 1–43.

[13] H. Keuning, J. Jeuring, and B. Heeren. 2018. A systematic literature review of automated feedback generation for programming exercises. *ACM Transactions on Computing Education* 19, 1 (2018). https://doi.org/10.1145/3231711 cited By 176.

[14] E. Kochmar, D.D. Vu, R. Belfer, V. Gupta, I.V. Serban, and J. Pineau. 2020. Automated Personalized Feedback Improves Learning Gains in An Intelligent Tutoring System. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 12164 LNAI (2020), 140–146. https://doi.org/10.1007/978-3-030-52240-7_26 cited By 24.

[15] E. Kochmar, D.D. Vu, R. Belfer, V. Gupta, I.V. Serban, and J. Pineau. 2022. Automated Data-Driven Generation of Personalized Pedagogical Interventions in Intelligent Tutoring Systems. *International Journal of Artificial Intelligence in Education* 32, 2 (2022), 323–349. https://doi.org/10.1007/s40593-021-00267-x cited By 15.

[16] X. Liu, H. Castellanos, L. Wiese, and A.J. Magana. 2023. Exploring Machine Learning Methods to Identify Patterns in Students' Solutions to Programming Assignments. *Proceedings - Frontiers in Education Conference, FIE* (2023). https://doi.org/10.1109/FIE58773.2023.10342972 cited By 0.

[17] Xiaojin Liu, Hugo Castellanos, Lucas Wiese, and Alejandra J Magana. 2023. Exploring Machine Learning Methods to Identify Patterns in Students' Solutions to Programming Assignments. In *2023 IEEE Frontiers in Education Conference (FIE)*. IEEE, 1–6.

[18] J. McBroom, I. Koprinska, and K. Yacef. 2022. A Survey of Automated Programming Hint Generation: The HINTS Framework. *Comput. Surveys* 54, 8 (2022). https://doi.org/10.1145/3469885 cited By 20.

[19] J. McBroom, K. Yacef, and I. Koprinska. 2020. Scalability in Online Computer Programming Education: Automated Techniques for Feedback, Evaluation and Equity. *Proceedings of the 13th International Conference on Educational Data Mining, EDM 2020* (2020), 802–805. https://www.scopus.com/inward/record.uri?eid=2-s2.0-85132115839&partnerID=40&md5=efb9629d71354344887d56117529b9e7 cited By 1.

[20] M. Messer. 2022. Grading Programming Assignments with an Automated Grading and Feedback Assistant. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 13356 LNCS (2022), 35–40. https://doi.org/10.1007/978-3-031-11647-6_6 cited By 0.

[21] M. Messer, N.C.C. Brown, M. Kölling, and M. Shi. 2023. Machine Learning-Based Automated Grading and Feedback Tools for Programming: A Meta-Analysis. *Annual Conference on Innovation and Technology in Computer Science Education, ITiCSE* 1 (2023), 491–497. https://doi.org/10.1145/3587102.3588822 cited By 4.

[22] M. Messer, 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, 1 (2024). https://doi.org/10.1145/3636515 cited By 3.

[23] Marcus Messer, Neil CC Brown, Michael Kölling, and Miaojing Shi. 2024. Automated grading and feedback tools for programming education: A systematic review. *ACM Transactions on Computing Education* 24, 1 (2024), 1–43.

[24] Marcus Messer, Neil C. C. Brown, Michael Kölling, and Miaojing Shi. 2024. Automated Grading and Feedback Tools for Programming Education: A Systematic Review. *ACM Trans. Comput. Educ.* 24, 1, Article 10 (feb 2024), 43 pages. https://doi.org/10.1145/3636515

[25] Fatema Nafa, Lakshmidevi Sreeramareddy, Sriharsha Mallapuram, and Paul Moulema. 2023. Improving Educational Outcomes: Developing and Assessing Grading System (ProGrader) for Programming Courses. In *International Conference on Human-Computer Interaction*. Springer, 322–342.

[26] J.W. Orr and N. Russell. 2021. Automatic Assessment of the Design Quality of Python Programs with Personalized Feedback. *Proceedings of the 14th International Conference on Educational Data Mining, EDM 2021* (2021), 495–501. https://www.scopus.com/inward/record.uri?eid=2-s2.0-85137237797&partnerID=40&md5=9c217bd17748b93a67fce110bbc70a07 cited By 7.

[27] M. Pankiewicz and R.S. Baker. 2023. Large Language Models (GPT) for automating feedback on programming assignments. *31st International Conference on Computers in Education, ICCE 2023 - Proceedings* 1 (2023), 68–77. https://www.scopus.com/inward/record.uri?eid=2-s2.0-85181532783&partnerID=40&md5=94b6a283167f03aee2752b7b2292ecb6 cited By 0.

[28] L. Roest, H. Keuning, and J. Jeuring. 2024. Next-Step Hint Generation for Introductory Programming Using Large Language Models. *ACM International Conference Proceeding Series* (2024), 144–153. https://doi.org/10.1145/3636243.3636259 cited By 0.

[29] Scott A Schartel. 2012. Giving feedback–An integral part of education. *Best practice & research Clinical anaesthesiology* 26, 1 (2012), 77–87.

[30] Thanveer Shaik, Xiaohui Tao, Yan Li, Christopher Dann, Jacquie McDonald, Petrea Redmond, and Linda Galligan. 2022. A review of the trends and challenges in adopting natural language processing methods for education feedback analysis. *IEEE Access* 10 (2022), 56720–56739.

[31] Diomidis Spinellis. 2006. *Code quality: the open source perspective*. Adobe Press.

[32] Xiaoyi Tian and Kristy Elizabeth Boyer. 2023. A Review of Digital Learning Environments for Teaching Natural Language Processing in K-12 Education. *arXiv preprint arXiv:2310.01603* (2023).

[33] C. Troussas, C. Papakostas, A. Krouska, P. Mylonas, and C. Sgouropoulou. 2023. Personalized Feedback Enhanced by Natural Language Processing in Intelligent Tutoring Systems. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 13891 LNCS (2023), 667–677. https://doi.org/10.1007/978-3-031-32883-1_58 cited By 8.

[34] Siyu Wu, Yang Cao, Jiajun Cui, Runze Li, Hong Qian, Bo Jiang, and Wei Zhang. 2024. A Comprehensive Exploration of Personalized Learning in Smart Education: From Student Modeling to Personalized Recommendations. *arXiv preprint arXiv:2402.01666* (2024).

[35] Yue Zhan and Michael S Hsiao. 2022. A Hybrid Approach for Automatic Feedback Generation in Natural Language Programming. In *2022 Fourth International Conference on Transdisciplinary AI (TransAI)*. IEEE, 32–39.

[36] J. Zhang, D. Li, J.C. Kolesar, H. Shi, and R. Piskac. 2022. Automated Feedback Generation for Competition-Level Code. *ACM International Conference Proceeding Series* (2022). https://doi.org/10.1145/3551349.3560425 cited By 1.

A AI USE DISCLOSURE

During the preparation of this work, the author(s) used ChatGPT-3.5, Copilot, Google Gemini and HuggingChat to generate feedback for the experiment in Section 6 on programming exercises for students to review and evaluate their usefulness, clarity, etc. Additionally, ChatGPT-4o was used to improve the clarity and grammar of the paper, making it easier to read. After using these tool/service, the author(s) reviewed and edited the content as needed and take(s) full responsibility for the content of the work.