Direct Automated Feedback Delivery for Student Submissions based on LLMs

Anonymous Author(s)

Abstract

Timely and individualized feedback is essential for students' learning progress and motivation, yet providing such feedback has become increasingly challenging due to growing student numbers. This has resulted in a time-consuming, repetitive, and often manual task for educators, contributing to a high workload.

This paper presents DAFeeD, an LLM-based approach for automated feedback on student submissions across various exercise domains. The defined feedback process enables interactive learning by allowing students to submit solutions multiple times and automatically receive iterative LLM feedback on their submission attempts before deadlines. By incorporating task details, grading criteria, student solutions, and custom instructions into the prompt, DAFeeD provides clear, personalized, and pedagogically meaningful feedback to support continuous improvement.

To evaluate the feedback process, we implemented DAFeeD in an open-source reference implementation integrated into the learning platform LP. A controlled study with students working on a programming task in a supervised environment showed that students found the feedback relevant and beneficial. They reported feeling more comfortable and willing to request automated feedback due to its convenience and immediacy. Additionally, deploying DAFeeD in a software engineering course with 450 students demonstrated improvements in student performance and encouraged iterative refinement through multiple submissions.

These findings highlight DAFeeD's potential to enhance feedback processes in computing education, improving both learning efficiency and student outcomes.

CCS Concepts

• **Social and professional topics → Student assessment**; • **Applied computing → Education**.

Keywords

Software Engineering, Education, Formative Feedback, Interactive Learning

ACM Reference Format:
Anonymous Author(s). 2024. Direct Automated Feedback Delivery for Student Submissions based on LLMs. In *Proceedings of 33rd ACM Symposium on the Foundations of Software Engineering (FSE '25).* ACM, New York, NY, USA, 11 pages. https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

In the current educational landscape, providing timely and effective feedback to students remains a significant challenge for both students and educators. Traditionally, students must wait for educators to review their submissions and provide feedback. This process can be time-consuming, often requiring students to arrange meetings and wait for available time slots, which are not always convenient or immediate. Similar it is time-consuming and tedious for educators to provide asynchronous feedback via email or other communication channels [14]. The inherent delays and scheduling difficulties make this approach not scalable, especially in courses with a large number of students.

Providing individualized feedback and enabling students to enhance their knowledge through formative assessments are important components of effective learning [15, 17]. However, the limited availability of educators means that not all students receive the individualized attention they need to improve their understanding and skills. This situation underscores the necessity for a more efficient and scalable feedback system that can provide continuous support and feedback to students without the constraints of traditional methods [31]. Such a feedback system enables interactive learning for students, increasing their engagement in the course resulting in better final grades [22].

In this paper, we present Direct Automated Feedback Delivery (DAFeeD), an approach for generating automated feedback on student submissions using the assistance of large language models (LLMs), to address these challenges. DAFeeD uses the exercise problem statement, predefined grading criteria (when available), the student solution and custom instructions to create a prompt for the LLM with the aim to create didactically meaningful feedback that supports the learning process. Designed to provide educators with full control and the ability to fine-tune the prompt, DAFeeD aims to reduce time spent on repetitive tasks, allowing educators to focus more on direct interactions with students. The approach is independent of the exercise type and can be applied and adapted to various domains, such as programming, text, or UML modeling exercises.

We implemented the approach in an open-source reference implementation called FeedbackSystem[1], connected to the learning platform LP[2] [6] through which students submit their solutions and receive feedback. To validate the effectiveness and efficiency of the approach, we first tested it in a controlled environment, then employed it in an actual software engineering course. With this paper, we want to answer the following research questions about direct automated feedback delivery:

RQ1 Do students feel more comfortable requesting automatic feedback from DAFeeD than asking a human educator?
RQ2 How do students perceive the effectiveness of DAFeeD?

[1]FeedbackSystem - Anonymized due to double-blind review regulations
[2]LP - Anonymized due to double-blind review regulations

RQ3 How do students perceive the usability and helpfulness of DAFeeD?
RQ4 Does the DAFeeD process improve students' performance?
RQ5 How does DAFeeD's feedback compare against human tutor feedback?

The subsequent sections of this paper are systematically structured to offer a thorough understanding of the research. Section 2 provides an overview of related work. Section 3 details the concept and methodology of DAFeeD. Section 4 describes the reference implementation of DAFeeD, called FeedbackSystem, including a general overview, details on the used prompts, and the system architecture. Section 5 describes the study design, presents the evaluation results, and outlines findings and limitations. Finally, Section 6 concludes with a summary of findings and discusses future research directions to enhance automated feedback systems.

2 Related Work

Automated feedback systems have gained significant attention in educational research due to their potential to scale online education and reduce the time between submission and feedback. Hahn et al. [13] conducted a systematic review on the effects of automatic scoring and feedback tools, emphasizing their crucial role in enhancing scalability, reducing bias, and increasing student engagement. Their insights highlight the broader implications of automated feedback systems in education, which is highly relevant to this work's study.

In the domain of programming education, Keuning et al. [20] reviewed 101 tools for automated feedback on programming exercises. They noted that most tools focus on error identification rather than providing actionable guidance or adapting to specific instructional needs. Extending this work, Kiesler et al. [21] explored the effectiveness of LLMs like ChatGPT in generating formative programming feedback, finding that while LLMs can produce useful feedback, they often include misleading information for novices. This emphasizes the need for careful design and evaluation of LLM-based feedback systems to ensure reliability and accuracy.

Kazemitabaar et al. [19] examined how novice programmers interact with LLM-based code generators in self-paced learning environments. They identified distinct usage patterns and their impact on learning outcomes, revealing that a "Hybrid" approach — combining manual coding with LLM assistance — was most beneficial for learners. This aligns with findings by Buçinca et al. [9], who highlighted the dangers of over-reliance on AI and proposed cognitive forcing functions to encourage deeper engagement with AI outputs. Similarly, Becker et al. [8] discussed both the opportunities and challenges of AI-driven code generation tools, emphasizing the need for educators to guide students in leveraging these technologies effectively without becoming dependent on them. These findings highlight the importance of balancing AI assistance with traditional learning methods, which is a key consideration in the DAFeeD approach.

The importance of timely and specific feedback is well-documented. Shute [30] provided a comprehensive review of formative feedback, highlighting its necessity for being nonevaluative, supportive, timely, and specific. Dawson et al. [11] further explored perceptions of effective feedback, revealing that while educators focus on design aspects such as timing and modalities, students prioritize the quality and usability of feedback comments. This underscores the need for automated feedback systems to deliver not only timely but also detailed, specific, and personalized comments.

Adaptive and immediate feedback mechanisms have been shown to significantly enhance student learning outcomes. Marwan et al. [26] demonstrated that adaptive and immediate feedback can improve student performance and motivation. Similarly, Leinonen et al. [23] compared immediate and scheduled feedback, concluding that immediate feedback is more effective in promoting student engagement and timely corrections. These studies collectively stress the potential of automated feedback systems in providing timely, adaptive, and engaging feedback, crucial for continuous improvement and learning efficiency.

The work by Azaiz et al. [7] highlights the limitations of LLMs, advising against using GPT-4 Turbo for automatic feedback generation in programming education due to inconsistencies. In contrast, this work's research with DAFeeD evaluates an integrated direct automatic feedback delivery process within a learning management system (LMS), demonstrating its potential immediate benefits for students, especially when feedback is critically evaluated. We believe that increasingly powerful LLMs and advanced prompting strategies will enhance feedback quality over time, with appropriate guardrails to prevent revealing solutions.

The study by Liffiton et al. [24] introduces CodeHelp, an LLM-powered tool that provides real-time assistance to programming students. In a first-year computer science course with 52 students, CodeHelp collected data over 12 weeks, revealing that students valued its availability, immediacy, and support for error resolution and independent learning. CodeHelp requires students to manually enter code, error messages, and issue descriptions. In contrast, DAFeeD integrates into the LMS, automatically providing context and feedback on code repository changes without requiring student input. This seamless integration aims to increase student engagement and motivation by offering timely, individualized feedback automatically, and to improve perceptions of feedback effectiveness, usability, and helpfulness.

Similarly, Nguyen and Allan [27] demonstrate the feasibility of using GPT-4 for tiered, formative feedback on programming exercises in introductory courses, providing insights on conceptual understanding, syntax, and time complexity. The DAFeeD approach proposed in this paper is evaluated using a similar LLM, GPT-4 Turbo, and likewise focuses on providing formative feedback in introductory courses. However, while Nguyen and Allan [27] provide feedback on isolated code snippets using few-shot learning, DAFeeD delivers iterative feedback on entire repositories with multiple files using detailed prompts and context collection. Additionally, DAFeeD is integrated directly into an LMS, supporting multiple exercise domains, which allows students to iteratively improve their submissions before the deadline, aiming to enhance learning outcomes and engagement through interactive learning.

Woodrow et al. [35] explore the deployment and effectiveness of a real-time style feedback tool using LLMs, specifically GPT-3.5 Turbo, in a large-scale online CS1 course. Their findings indicate significant improvements in student engagement and coding style when feedback is immediate and integrated within the learning platform. Woodrow et al. conducted a randomized control trial with over 8,000 students, demonstrating that real-time feedback was five times more likely to be viewed and incorporated by students compared to delayed feedback. This supports the approach with DAFeeD, emphasizing the importance of immediate, individualized feedback in enhancing student learning outcomes.

Besides programming exercises, recent studies have explored using LLMs to provide automated feedback in UML modeling education. Ardimento et al. [4] introduced a cloud-based tool that utilizes Retrieval-Augmented Generation (RAG) to analyze UML diagrams and offer contextually relevant suggestions. An evaluation with 5,120 labeled UML models demonstrated its effectiveness in helping students identify and correct common errors, enhancing their understanding of UML concepts. A follow-up study further improved error detection and provided real-time, constructive feedback, with user feedback highlighting the tool's potential to enhance software modeling education [5].

Similarly, Cámara et al. [10] examined generative AI tools, such as ChatGPT, for formative assessment in UML modeling tasks. Their findings show that AI-assisted feedback helps students track their progress and improve performance compared to traditional methods. However, the study also emphasizes the importance of educating students on AI limitations to prevent over-reliance on automated feedback.

A broader perspective on the capabilities and challenges of LLMs is provided by Wei et al. [32], who discussed the emergent abilities of LLMs that are not present in smaller models, highlighting the need for ongoing research to harness these capabilities effectively. Huang et al. [16] addressed the issue of hallucinations in LLMs, offering an in-depth overview of detection methods and mitigation strategies crucial for developing reliable feedback systems. Amatriain [3] detailed core concepts and advanced techniques in prompt engineering, such as Chain-of-Thought and Reflection, which can enhance the quality and relevance of automated feedback. Additionally, Liu et al. [25] highlighted the importance of robust prompt design to prevent misuse, investigating vulnerabilities of LLMs to jailbreak prompts. Zhao et al. [38] reviewed the evolution and recent advances of LLMs, focusing on pre-training, adaptation tuning, utilization, and capacity evaluation, and highlighting the progress and ongoing challenges in the field. Lastly, Yang et al. [36] provided a comprehensive guide for practitioners working with LLMs, discussing the influence of pre-training data and challenges associated with different natural language processing tasks, offering insights for developing and deploying LLM-based feedback systems.

In summary, the related work collectively highlights the evolving landscape of automated feedback systems and the significant potential of LLMs to enhance educational outcomes through immediate, specific, and actionable feedback. The primary contribution with DAFeeD lies in its seamless integration within the LMS, supporting a variety of exercise types and having access to all relevant context information used for the feedback prompt. By using LLMs, DAFeeD provides individualized feedback automatically, with a focus on the feedback delivery process. This approach enables students to iteratively improve their solutions and learn continuously without direct intervention from tutors or professors. All of this is possible without leaving their preferred learning platform. DAFeeD aims to enhance student engagement, learning efficiency, and performance through timely, relevant, and personalized feedback, aligning with and advancing the findings of the reviewed studies.

3 Approach: Direct Automated Feedback Delivery (DAFeeD)

To complement traditional teaching methods and provide additional support, DAFeeD employs LLMs to deliver automated feedback on student submissions. Figure 1 illustrates the continuous feedback workflow that DAFeeD facilitates, enabling students to receive feedback at any time, thereby eliminating the need to wait for responses from human educators.

The feedback process is designed to be exercise-independent, meaning that it can be applied and adapted to various exercise types, such as programming, text, or modeling exercises. DAFeeD can automatically provide feedback to the students, including feedback on issues or improvements, as well as positive feedback when the student completes the task correctly. Once the student submits their solution, DAFeeD initiates a three-stage process to generate natural language feedback.

The first stage, called *Formatting*, takes the student's submission and extracts the submission content, the problem statement, including learning objectives, and any possible grading instructions the educator defines. In addition, the student's learner profile [1] is considered to generate individualized, personal feedback tailored to the student's skills and learning styles. All of this gathered information represents the context.

During the prompt generation step, a predefined prompt template is filled with the individual prompt input data that is included in the context, resulting in the feedback prompt. Depending on the exercise, adaptions need to be made to the prompt template to ensure that the feedback output of the LLM is tailored to the specific exercise type. For programming exercises, the generated feedback needs to have metadata information about the file and line number of the code snippet to which the feedback refers. In the case of text exercises, the feedback needs to contain metadata identifying the specific sentence or word range to which it applies. Similarly, for modeling exercises, metadata must reference the corresponding model element or relation to ensure precise feedback alignment.

In the second stage, called *Predicting*, DAFeeD sends the feedback prompt to a LLM and invokes it with the prompt. As a result, the LLM generates a response to that prompt including detailed feedback items for the student.

The final stage, *Parsing*, takes the LLM response, which comes in the JSON format, and parses feedback items from it. In addition to the feedback text, the feedback object also contains reference information indicating the part of the submission it pertains to. For programming exercises, this includes the file name and line number of the relevant code snippet to which the feedback refers. For text exercises, the reference information includes only the corresponding sentence or word range. When it comes to modeling exercises, the feedback needs to reference the specific model element or relation it pertains to.

All of the feedback is then returned to the student for review. If the student is satisfied with the feedback, the process concludes. Otherwise, the student can refine and resubmit their solution, initiating the DAFeeD process anew.

This iterative process is designed to motivate students to continuously learn and experiment with their solutions, resulting in improved performance.

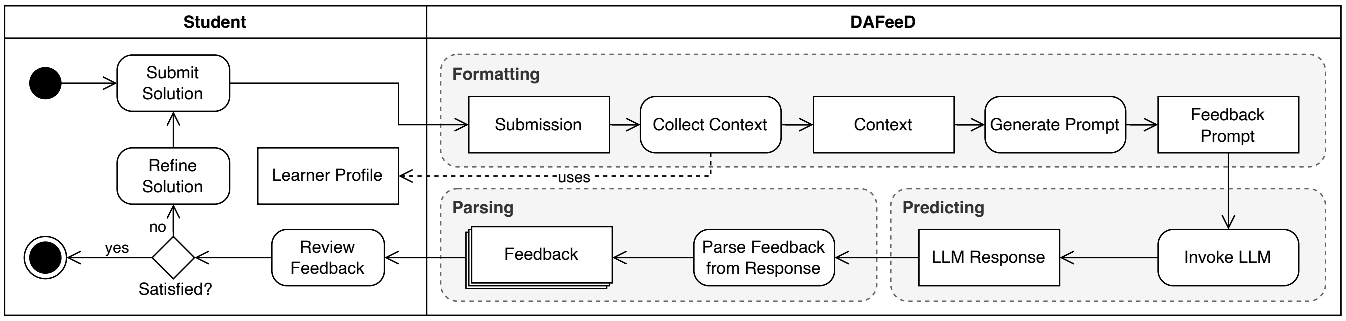
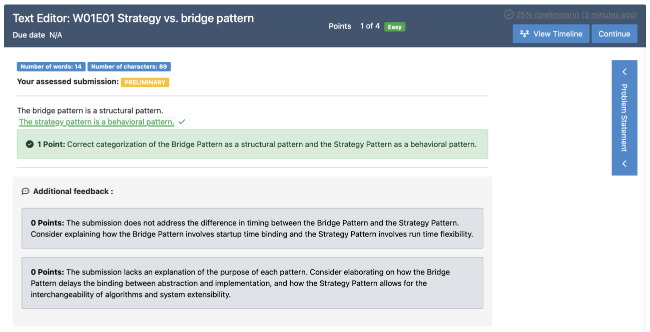


Figure 1: UML activity diagram of the Direct Automated Feedback Delivery (DAFeeD) workflow for student submissions.

4 Reference Implementation: FeedbackSystem

We incorporated DAFeeD into a reference implementation named FeedbackSystem, which is seamlessly integrated with the learning platform LP. Through LP, students can submit their solution and review the feedback. FeedbackSystem supports the feedback generation for programming, text, and modeling exercises.

When submitting their solutions on LP, students have the option to request direct automated feedback by clicking a dedicated button. This feedback request is then sent to FeedbackSystem, assuming the student has not reached their feedback request limit for the exercise. Educators can customize the limit of allowed feedback requests per exercise according to their preference. A status visualization informs students about their feedback request state. Once FeedbackSystem generates the feedback and sends it back to LP, the student can review it in an inline feedback view window on LP. The feedback view is tailored to different exercise types, ensuring that feedback is presented in a format most suitable for the nature of the task. For instance, text exercises include detailed inline comments, while modeling exercises feature visual annotations on model elements and relations. An example visualization of the inline feedback interface for a text exercise submission is depicted in Figure 2.


Text Editor: W01E01 Strategy vs. bridge pattern
Due date N/A
Points 1 of 4 Easy
25% (preliminary) (3 minutes ago)
View Timeline
Continue
Problem Statement

Number of words: 14 Number of characters: 89

Your assessed submission: PRELIMINARY

The bridge pattern is a structural pattern.
The strategy pattern is a behavioral pattern. ✓

1 Point: Correct categorization of the Bridge Pattern as a structural pattern and the Strategy Pattern as a behavioral pattern.

Additional feedback :

0 Points: The submission does not address the difference in timing between the Bridge Pattern and the Strategy Pattern. Consider explaining how the Bridge Pattern involves startup time binding and the Strategy Pattern involves run time flexibility.

0 Points: The submission lacks an explanation of the purpose of each pattern. Consider elaborating on how the Bridge Pattern delays the binding between abstraction and implementation, and how the Strategy Pattern allows for the interchangeability of algorithms and system extensibility.

Figure 2: Visualization of the inline feedback interface for text exercises in the LP as seen by students.

The inline feedback view for text exercises offers a targeted approach by directly associating feedback items with specific text spans, such as highlighting the second sentence in the example shown. The interface also distinguishes between referenced feedback, tied to specific text, and unreferenced feedback, offering general observations.

4.1 Feedback Generation

The prompt design is crucial for guiding the LLM in generating effective and contextually relevant feedback. The system is configurable, allowing the use of different LLMs and model settings. In Listing 1, we provide an example of a prompt used for generating feedback for programming exercises. This prompt incorporates specific instructions to ensure that the feedback is individualized to the student's submission while not revealing the solution.

The feedback generation process for programming exercises begins by identifying the differences between the student's submission repository and the provided template repository. These differences are identified using a git diff, which highlights the lines removed and added by the student. If the problem statement is too lengthy or complex, a separate LLM invocation is used to split the problem statement into relevant parts for each file. This ensures that the feedback is targeted and relevant to the specific context of the file being reviewed. Additionally, a summary of the student's solution across all files is generated using another LLM invocation. This summary provides a comprehensive overview of the submission, which is included in the prompt to offer context for the feedback.

In the provided prompt, several key components guide the LLM in creating useful feedback. The *Problem Statement* section contextualizes the student's task and helps the LLM understand the exercise's objectives. The *Task Instructions* direct the LLM to provide improvement suggestions focusing on educational aspects without offering direct solutions. *Style Guidelines* ensure the feedback is constructive, specific, balanced, clear, concise, actionable, educational, and contextual. The *File Path and Content* provide the specific file under review along with its content, aiding the LLM in pinpointing specific lines of code for feedback. Additionally, *Summary and Diffs* between the template and submission offer additional context, helping the LLM understand the student's changes and their overall approach.

In contrast to other external AI tools such as ChatGPT, FeedbackSystem is integrated directly into the LP and has direct access to every relevant context information, providing immediate feedback to students without requiring manual input. With FeedbackSystem in place, students do not need to switch between different windows and do not need to copy-paste their solution or the problem statement into other tools. FeedbackSystem is also designed in a way that it provides didactically sensible feedback without giving away the solution, which is crucial for educational purposes.

The structure and content of this prompt are designed to emulate a human tutor's approach, ensuring that the feedback is both relevant and supportive of the student's learning process. By providing such detailed instructions and contextual information, the LLM can generate feedback that is both useful and actionable for students.

```
You are an AI tutor for programming exercises at a prestigious university.

# Problem statement
{problem_statement}

# Task
Create non graded improvement suggestions for a student's programming
submission that a human tutor would recommend. Assume the tutor is not familiar
with the solution. The feedback must contain only the feedback the student can
learn from. Important: the answer you generate must not contain any solution
suggestions or contain corrected errors. Rather concentrate on incorrectly applied
principles or inconsistencies. Students can move some functionality to other files.
Students can deviate to some degree from the problem statement or book unless
they complete all tasks. Very important, the feedback must be balanced.

# Style
1. Constructive, 2. Specific, 3. Balanced, 4. Clear and Concise, 5. Actionable, 6.
Educational, 7. Contextual

It is strictly prohibited to include feedback that contradicts to the problem
statement.
No need to mention anything that is not explicitly in the template->submission diff,
 as it is out of student's control (e.g. exercise package name).

In git diff, lines marked with '-' were removed and with '+' were added by the
student.

# The student will be reading your response, use you instead of them

Path: {submission_file_path}

File (with line numbers <number>: <line>):
{submission_file_content}

Summary of other files in the solution:
{summary}

The template to submission diff (only as reference):
{template_to_submission_diff}
```

Listing 1: Prompt template for generating feedback for programming exercises.

4.2 Architecture

FeedbackSystem is deployed in production alongside the learning platform LP, which serves up to 2000 students per course. Consequently, the reference implementation must satisfy additional non-functional requirements such as performance, scalability, maintainability, and usability. To meet these requirements and to support feedback generation for multiple exercise types while allowing for future extensibility, we adopted a modular architecture, as illustrated in Figure 3.

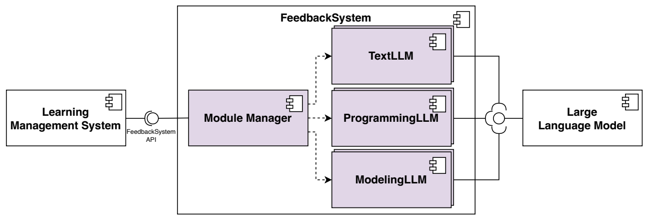


Figure 3: UML component diagram showing the top-level architecture of the reference implementation.

The *Module Manager* handles all incoming requests, verifies authorization, and forwards them to the appropriate modules. The *ProgrammingLLM* module manages programming exercises and executes the three-stage DAFeeD process, which includes formatting, predicting, and parsing. Similarly, the *TextLLM* and *ModelingLLM* module are optimized for text and modeling exercises, respectively, and follow the same process.

FeedbackSystem's system design is independent of any specific learning management system (LMS) as it provides a REST API, documented using the OpenAPI standard[3]. This independence allows FeedbackSystem to be integrated with various LMS platforms, such as Moodle[4], bringing the benefits of DAFeeD to more universities and students.

FeedbackSystem currently connects to OpenAI models hosted in a private Azure cloud to ensure that student data is not used for training models, maintaining privacy. Additionally, the system can be configured to use open-source models like Llama[5] or Mistral[6], either self-hosted or cloud-based.

To meet performance and scalability requirements, the FeedbackSystem and its modules are deployed within a Kubernetes cluster[7]. Kubernetes, in conjunction with the FeedbackSystem's modular architecture, allows the system to scale each module independently. For example, additional instances of the programming module can be instantiated when a new programming exercise is released to handle the increased load. Furthermore, Kubernetes provides out-of-the-box load balancing to distribute the load between multiple module instances and self-healing capabilities, ensuring that if a module crashes, it gets automatically restarted.

5 Evaluation

In this section, we outline the methodology employed to validate the effectiveness of the proposed DAFeeD approach including the reference implementation FeedbackSystem. The conducted evaluation represents the treatment validation phase of the design science methodology proposed by Wieringa [34]. For this phase, we first evaluated the proposed solution — DAFeeD — in a controlled environment followed by a field study. The collected data is then utilized for the refinement and improvement of the solution.

We begin by describing the study design and the results. Subsequently, we outline the limitations of the evaluation and discuss the implications of the findings.

5.1 Study Design

The evaluation of the DAFeeD approach was conducted in two distinct stages to ensure a comprehensive assessment of its effectiveness. The first stage involved a controlled lab experiment where selected students interacted with the system and subsequently provided their perceptions through a structured survey. This stage aimed to capture initial user impressions and identify potential usability issues. In the second stage, DAFeeD and its reference implementation, FeedbackSystem, were deployed in an advanced software engineering course on design patterns. During this stage,

[3]https://www.openapis.org
[4]https://moodle.org
[5]https://llama.meta.com
[6]https://mistral.ai
[7]https://kubernetes.io

quantitative data was collected to evaluate both student performance and system effectiveness in a real-world educational setting.

For the first stage, we designed the "Code Review" Java programming exercise, a past introduction to software engineering homework assignment, to simulate a real-world scenario where students review and improve existing code. The exercise included tasks such as improving Java classes by following good coding practices, refactoring duplicated code using the template method design pattern, catching edge cases in functions, and implementing forgotten methods in the service package. We enabled FeedbackSystem for this exercise, utilizing OpenAI's GPT-4 Turbo with a temperature setting of 0 to ensure deterministic and consistent feedback generation, reducing randomness and the potential for hallucinations.

We invited 20 participants from current courses at the university via direct messages, including undergraduate and graduate students from various disciplines like computer science, information systems, and games engineering. Participants received a two-page participation manual and tested the new feedback feature on the LP in a controlled university environment. The evaluation lasted around 45 minutes, focusing on understanding the feedback process rather than completing the exercise. Participants followed a structured procedure, illustrated in Figure 4. They started by thoroughly reading the participation manual. Next, they prepared their IDE and accessed the exercise on LP. They then worked on the exercise, committed and pushed their code, and requested AI feedback iteratively. After reviewing and acting on the AI-provided feedback, they refined their solutions until they had a good understanding of the feedback process.

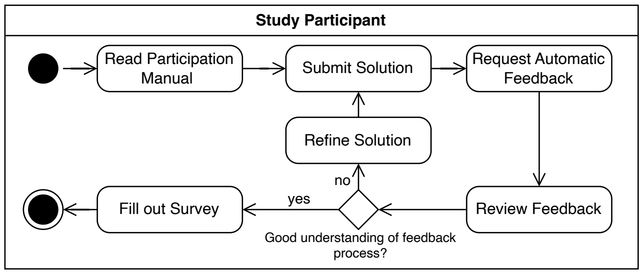

Study Participant
Read Participation Manual
Submit Solution
Request Automatic Feedback
Refine Solution
no
yes
Fill out Survey
Review Feedback
Good understanding of feedback process?

Figure 4: UML activity diagram illustrating the study procedure from a participant's perspective.

Following this hands-on experience, participants were asked to complete a survey hosted on the community version of the open-source survey tool LimeSurvey[8]. This survey aimed to gather their opinions on direct automated feedback and collect feedback on their overall experience with the feature. The study employed a mixed methods approach, combining quantitative and qualitative data collection methods.

All survey questions, except for the introductory demographic queries and five final voluntary free-text responses, employed a 5-point Likert scale [2] ranging from "strongly agree" to "strongly disagree" and were mandatory. The survey questions mapped to the research questions as follows:

RQ1 Comfort with Feedback Source
- **Q1** I feel more comfortable requesting direct automated feedback than feedback from a human tutor.
- **Q2** I am likely to request feedback more frequently when using direct automated feedback than feedback from my course professor.
- **Q3** I find receiving direct automated feedback less intimidating than receiving feedback from a human tutor.
- **Q4** I feel that requesting direct automated feedback is more convenient than arranging a meeting with a human tutor.

RQ2 Perceived Effectiveness
- **Q5** The direct automated feedback helps me understand my mistakes.
- **Q6** The direct automated feedback is more effective than one-time feedback.
- **Q7** The direct automated feedback has significantly improved the quality of my programming assignment.
- **Q8** The direct automated feedback is a helpful addition to the automatic test case results.
- **Q9** I feel that having access to direct automated feedback continuously helps me more than arranging a meeting with a human tutor.

RQ3 Usability and Helpfulness
- **Q10** It is easy to receive direct automated feedback on my programming assignments.
- **Q11** I would rather use the direct automated feedback integrated into LP than use an external AI tool for getting feedback.
- **Q12** I find the direct automated feedback helpful in improving my programming skills.
- **Q13** I am satisfied with the overall performance of the direct automated feedback.

Following the initial lab experiment and survey, the second stage of the evaluation involved deploying the FeedbackSystem in the *Software Design Patterns* course, an advanced software engineering lecture with more than 500 active bachelor's and master's students. This stage aimed to assess the system's performance and scalability in a real-world educational setting. The deployment allowed for an in-depth investigation of the system's impact on student performance (RQ4) and how it compares to human feedback (RQ5).

The FeedbackSystem, configured with OpenAI's GPT-4o LLM, was enabled for two exercises, a modeling exercise and a text exercise. In the text exercise, students had to explain the difference between the strategy and the bridge design pattern, and could achieve a maximum of 4 points. The modeling exercise required students to create a UML class diagram of a car rental system and was worth 7 points.

Students could submit their solutions and iteratively request automated feedback to improve their work. After the exercises' deadline, which was one week after the release, all submissions were assessed by human tutors to determine the final scores. Based on the usage data collected during this stage, we analyzed student improvement over multiple submission iterations and also compared the feedback scores from FeedbackSystem against the human tutors' assessments.

5.2 Results

In the following paragraphs, we present the results, starting with stage one, the controlled lab experiment, and then moving on to

[8]https://www.limesurvey.org

stage two, the deployment in the advanced software engineering course. The answers to each of the Likert scale questions are visualized in Figure 5.

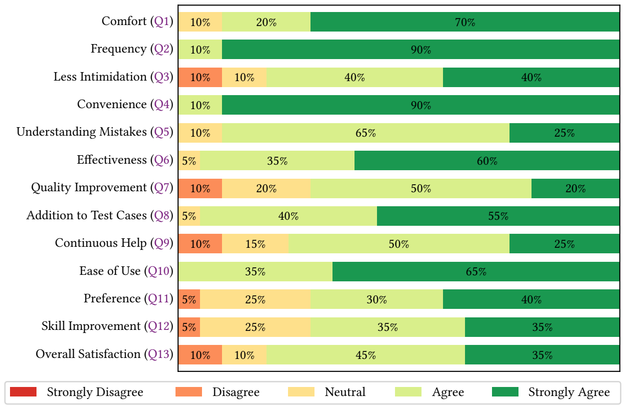


Figure 5: Distribution of survey responses to Likert scale questions.

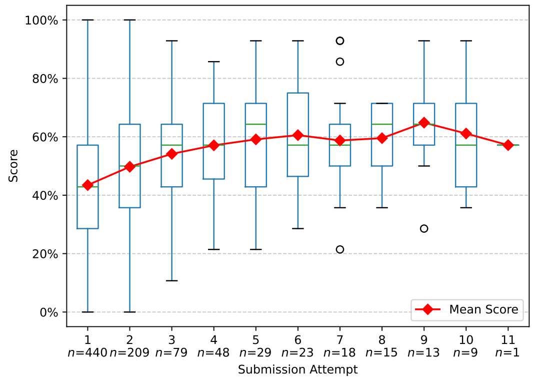
Comfort levels in requesting feedback showed that 90% of students feel more comfortable requesting direct automated feedback from LP than feedback from a human tutor, with 10% neutral (Q1). For Q2, 100% of students noted that they are likely to request feedback more frequently when using direct automated feedback than feedback from their course professor. Receiving feedback from LP was found to be less intimidating than from a human tutor by 80% of students, with 10% neutral and 10% disagreeing (Q3). Convenience in requesting feedback showed that 100% of students feel that requesting direct automated feedback is more convenient than arranging a meeting with a human tutor (Q4).

In terms of understanding mistakes, 90% of students believed that the direct automated feedback provided by LP helps them understand their mistakes, with 10% neutral (Q5). The effectiveness of the feedback was highlighted by 95% of students who found that the direct automated feedback is more effective than one-time feedback, with 5% neutral (Q6). Regarding the quality of assignments, 70% of students observed that DAFeeD has significantly improved the quality of their programming assignments, with 20% neutral and 10% disagreeing (Q7). For Q8, 95% of students felt that the direct automated feedback is a helpful addition to the automatic test case results, with 5% neutral. Continuous access to feedback was found to be more beneficial than arranging meetings with a tutor by 75% of students, with 15% neutral and 10% disagreeing (Q9).

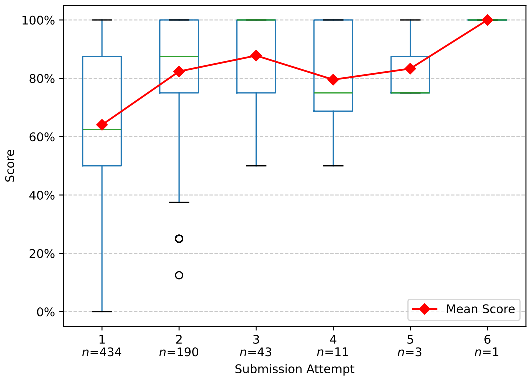
Ease of receiving feedback was highly rated, with 100% of students confirming that it is easy to receive direct automated feedback on their programming assignments (Q10). Furthermore, 70% of students preferred using DAFeeD integrated into LP over using an external AI tool for getting feedback, with 25% neutral and 5% disagreeing (Q11). In terms of skill improvement, 70% of students agreed that they find the direct automated feedback helpful in improving their programming skills, with 25% neutral and 5% disagreeing (Q12). Lastly, 80% of students were satisfied with the overall performance of DAFeeD, with 10% neutral and 10% disagreeing (Q13).

The responses to the five voluntary free text questions highlight several themes. Many students appreciated the immediate availability of feedback, which allowed for prompt corrections without waiting for manual review. However, some respondents suggested improvements such as better categorization of feedback, more detailed explanations of errors, and prioritization of critical issues. The feedback was generally found to be relevant and useful in addressing obvious mistakes and improving code quality. Students expressed a preference for feedback that clearly identified mistakes and provided specific guidance on how to correct them, along with suggestions for improvement. Some challenges included understanding certain automated feedback messages and occasional false positives or negatives in error detection.

For stage two we first analyzed the distribution of scores across submission attempts for the modeling and text exercises, as shown in Figure 6. Since students were not obliged to request automatic feedback, the sample sizes differ across submission attempts. The results indicate an overall improvement in scores with subsequent submission attempts.



(a) Score distribution for the modeling exercise.



(b) Score distribution for the text exercise.

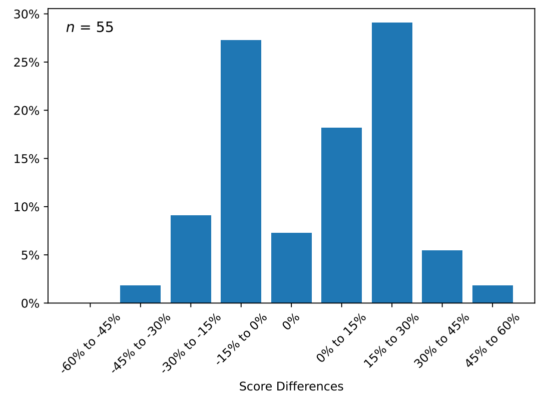
Figure 6: Distribution of FeedbackSystem's scores across submission attempts.

In the modeling exercise (Figure 6a), the average scores increased steadily across multiple submission attempts. The initial average score was 44% in the first attempt, rising to 61% by the sixth attempt. However, a slight decline was observed in the seventh attempt, where the score dropped to 59%. The scores then showed a peak of 65% in the ninth attempt. After this peak, the scores declined down to 57% in the eleventh attempt.

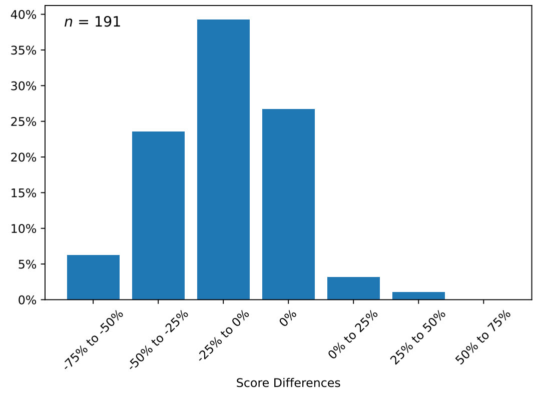
For the text exercise (Figure 6b), the average scores exhibited a more consistent upward trajectory. The initial average score of 64% in the first attempt increased significantly to 82% by the second attempt and further improved to 88% by the third attempt. Despite a minor decline to 80% in the fourth attempt, subsequent submissions saw an upward trend from 83% in the fifth to 100% in the sixth attempt.

We analyzed the score differences between the FeedbackSystem and human tutors for both exercises, as illustrated in Figure 7. To ensure a fair comparison, only the latest student submissions that received feedback from the FeedbackSystem and remained unchanged until the exercise deadline were considered. This approach guarantees that both the FeedbackSystem and human tutors assessed identical versions of student submissions.

The score differences were computed by subtracting the FeedbackSystem's assigned score from the human tutor's score, where negative values indicate instances in which the human tutor assigned a lower score than the FeedbackSystem. To facilitate the visualization of a 1-point score deviation, the data was grouped into equal-sized intervals based on the total achievable points for each exercise, with each interval representing 15% (1/7) of the total score for the modeling exercise and 25% (1/4) for the text exercise.



(a) Score differences for the modeling exercise.



(b) Score differences for the text exercise.

Figure 7: Distribution of score differences between the FeedbackSystem and a human tutor. Negative values indicate a lower score from the tutor.

The analysis of score differences between the FeedbackSystem and the human tutors revealed distinct trends for the modeling and text exercises. For the modeling exercise, the FeedbackSystem generally assigned lower scores compared to the human tutors, as visualized in Figure 7a. 29% of submissions received scores that were between 15% and 30% higher when graded by the human tutors. Additionally, 18% had smaller underestimations, with differences ranging from 0% to 15%. In contrast, the FeedbackSystem provided higher scores than the human tutors for 27% of submissions, with differences ranging from -15% to 0%. A moderate overestimation was observed in five submissions (9%), with score differences between -30% and -15%, while extreme overestimations were rare (2%) falling within the -45% to -30% range. 7% of submissions received identical scores.

For the text exercise, the FeedbackSystem consistently provided higher scores than the human tutors, as depicted in Figure 7b. The majority of submissions (39%) fell within the range of -25% to 0%, indicating that the FeedbackSystem assigned slightly higher scores. A significant portion (24%) showed more considerable overestimations within the -50% to -25% range, while 6% experienced substantial overestimations ranging from -75% to -50%. In contrast, identical scores were observed for 51 submissions (27%). Cases where the FeedbackSystem assigned lower scores than the human tutors were infrequent, with only 4% showing positive differences, and no submissions exceeded a difference of 50%.

5.3 Findings

The responses to RQ1 reveal a strong level of comfort for requesting automated feedback compared to traditional human feedback channels. Students stated they feel more comfortable requesting automated feedback than from human tutors and were likely to request automated feedback more frequently than from their course professors. Requesting automated feedback was perceived as less intimidating for most of the participants and all of them stated it is more convenient than arranging meetings with a human tutor. These findings highlight the effectiveness of automated feedback in providing a more comfortable and accessible feedback mechanism for students.

Main Findings for RQ1: Students feel more comfortable requesting automated feedback than human feedback. They are likely to request automated feedback more frequently and find it less intimidating and more convenient than arranging meetings with a human tutor.

The responses to RQ2 indicate that students think automated feedback is highly effective in helping them understand and improve their programming assignments. Students reported that the feedback helped them understand their mistakes and found it more effective than receiving only one-time feedback for their submission. The majority reported that the feedback significantly improved the quality of their programming assignments, and all participants stated that automatic feedback is a helpful addition to automatic test case results generated by LP. In addition, most participants saw continuous access to automated feedback as more beneficial than arranging meetings with a tutor. These findings suggest that automated feedback not only aids in error identification but also significantly enhances the overall quality of student assignments.

Main Findings for RQ2: Students perceive automated feedback as highly effective in helping them understand and improve their programming assignments. The feedback helps them understand their mistakes, improves the quality of their assignments, and is a helpful addition to automatic test case results.

The responses to RQ3 demonstrate the ease of receiving feedback and overall satisfaction with DAFeeD's feedback process and its reference implementation. Students found it easy to receive feedback on their programming assignments. A large number of participants preferred using the feedback integrated into LP than copy their submission and relevant context information over to an external AI tool. Most participants also deemed the feedback helpful in improving their programming skills. In regard to the overall performance of DAFeeD, the majority of students expressed high satisfaction with the system.

Main Findings for RQ3: Students find it easy to receive feedback on their programming assignments and are satisfied with the overall performance of DAFeeD and its reference implementation. There are some suggestions for improvements, such as better categorization of feedback, more detailed explanations of errors, and prioritization of critical issues.

The analysis of the score distributions across submission attempts for the modeling and text exercises revealed an overall improvement in student performance over time. Average scores increased steadily during the initial attempts, followed by a slight decline in later attempts. This decline can be attributed to students who achieved high scores early being less likely to submit further, leaving primarily lower-scoring students to continue. The decreasing number of submissions over time supports this explanation, indicating that students who met their goals early were less inclined to make additional attempts. The observed improvements suggest that the direct automated feedback provided by the FeedbackSystem effectively supports student learning by guiding them through an iterative improvement process.

Main Findings for RQ4: Students improved their performance across multiple submission attempts with the feedback provided by the FeedbackSystem. The steady increase in scores indicates the effectiveness of automated feedback in facilitating learning.

The comparison of scores between the FeedbackSystem and human tutors for the modeling and text exercises revealed distinct trends. In the modeling exercise, the FeedbackSystem generally assigned lower scores compared to human tutors, suggesting a more conservative grading approach or limitations in the system's ability to fully interpret complex submissions. Conversely, in the text exercise, the FeedbackSystem tended to provide higher scores compared to human tutors. Despite these trends, the analysis showed that over 52% of modeling submissions and more than 69% of text submissions exhibited either no difference or a 1-point difference between the FeedbackSystem and human tutors. These results suggest that the quality of the feedback provided by the FeedbackSystem is comparable to that of human tutors, although some discrepancies remain and present opportunities for future improvement. Furthermore, we noticed the critical role of well-defined grading criteria in enhancing the quality of automated feedback. Similar to manual assessment by human tutors, clear and course-specific grading criteria are essential for ensuring consistency and relevance in automated evaluations.

Main Findings for RQ5: The FeedbackSystem generally assigned lower scores in modeling exercises and higher scores in text exercises. Over half of the submissions showed minimal score differences, indicating comparable feedback quality.

5.4 Discussion

Overall, students received DAFeeD's reference implementation FeedbackSystem integrated into LP positively, indicating its effectiveness in enhancing their skills. The iterative submission process, supported by automated feedback, allowed students to refine their work progressively, leading to measurable improvements in their performance across multiple submission attempts.

The findings validate the role of automated feedback in fostering an interactive learning environment. Students reported feeling more comfortable requesting automated feedback than human feedback, highlighting the system's convenience and accessibility. This interactive aspect of learning, facilitated by timely and continuous feedback, helps students develop critical problem-solving skills and deepen their understanding of programming concepts.

Despite the overall positive reception, there is room for improving the quality of the feedback provided. The comparative analysis of FeedbackSystem and human tutor scores highlighted certain discrepancies, with the system generally assigning lower scores in modeling exercises and higher scores in text exercises. While over half of the submissions showed only minimal differences between the automated and human evaluations, the remaining inconsistencies suggest that refinements are needed to better align the automated scoring with human judgment. Here it is important to note that while tutor assessments are treated as ground truth data in this analysis, but human grading is not infallible. The approach presented in this paper primarily supports the concept of formative assessment and is particularly effective when implemented fully automatically, especially in contexts where summative assessment is not the primary focus.

While students found the feedback helpful in understanding their mistakes and improving their assignments, we need to further refine it to enhance its precision and relevance. Incorporating more advanced AI techniques, such as Chain-of-Thought [33], ReAct [37], Retrieval-Augmented Generation (RAG) [4, 12], could address these issues, offering more tailored and context-specific guidance to students. Specifically, a RAG approach could enhance FeedbackSystem by retrieving relevant course materials, such as lecture slides, course discussions, and past feedback examples, to provide more contextually accurate and tailored responses that align with the course's learning objectives, offering students clearer explanations and actionable suggestions for improvement.

This work aligns with findings from other studies, highlighting the importance of timely and continuous feedback in education [11, 30]. Additionally, we believe that direct automated feedback should complement, rather than substitute, traditional human feedback. The convenience and accessibility of automated feedback are clear advantages, yet the nuanced insights that human tutors provide remain irreplaceable, including emotional support and empathy.

As the study is intended as the treatment validation stage for the proposed feedback process and should also provide important insights into the usability of the reference implementation, we assume that the results correctly reflect the tendency despite the rather small sample size for the controlled lab experiment. Nielsen [28] has shown that smaller, iterative tests can reveal the majority of usability issues, making such an approach both practical and effective.

We acknowledge that different challenges exist when using AI in education, and we are aware of their potential to negatively influence the learning experience of the diverse student body [18]. First, the competent use of AI must be explicitly taught to both educators and students to ensure effective and ethical utilization. Addressing the issue of AI hallucinations, a RAG approach can be employed to enhance accuracy [12]. To mitigate over-reliance on AI, it is crucial to encourage students to independently solve certain problems, fostering critical thinking and problem-solving skills. Bias in AI can be reduced by using models that are designed to minimize prejudices, ensuring fairness in feedback. Additionally, privacy and cost concerns can be alleviated by deploying locally trained open-source models, which offer greater control over data and reduce dependency on commercial AI solutions. These measures collectively promote a balanced and responsible integration of AI in the educational process.

In conclusion, DAFeeD shows significant potential in improving computing education. By addressing the areas identified for improvement and continuing to refine the system, we can harness the full benefits of automated feedback to support and enhance student learning.

5.5 Limitations

We follow the categorization framework proposed by Runeson and Höst [29] to outline the limitations of the conducted evaluation, addressing potential threats to internal, external, and construct validity:

Internal Validity: This study may be compromised by using self-reported survey data, which can introduce biases. Participants' perceptions may be influenced by their individual attitudes or varying levels of familiarity with programming concepts, leading to potential inaccuracies. Additionally, the participants' perceived effectiveness does not necessarily correspond to objective effectiveness. The assumption that human tutor assessments are entirely accurate may affect the validity of our findings, as tutors can introduce subjective biases or errors in their evaluations.

External Validity: Threats to external validity arise from the specific context of this study. Conducting the research exclusively at a single university and with students from computer science, information systems, and similar programs restricts the diversity of the sample. This narrow focus may limit the applicability of the findings to other educational settings or student populations. In addition, the small sample size of the controlled lab experiment may also limit the generalizability of some findings.

Construct Validity: The survey questions designed to evaluate 'perceived effectiveness' and 'comfort with feedback source' may not fully encompass the breadth of these constructs. Factors such as prior experiences and personal preferences, which the survey does not account for, could influence participants' responses and perceptions. The use of a specific sample exercise to evaluate the automated feedback process may align more closely with some students' prior experiences or learning styles, introducing bias into the results.

6 Conclusion

The main contributions of this paper include the introduction of DAFeeD, a direct automated feedback delivery system, and the development of its reference implementation, FeedbackSystem, which integrates DAFeeD into the learning platform LP. In addition, we conducted a comprehensive evaluation to demonstrate the system's effectiveness in improving student performance and engagement through iterative feedback. DAFeeD enables the interactive learning process by providing students with immediate, context-specific feedback on their submissions. The implementation of DAFeeD demonstrates the feasibility and advantages of incorporating automated feedback systems into computing education. For educators, this can alleviate the heavy workload associated with providing individualized feedback, allowing them to focus more on instructional design and student support. For students, the evaluation of DAFeeD showed that they perceive the automated feedback as effective, helpful, and easy to use, and that it enhances their engagement and motivation.

Future work includes enhancing the visualization of feedback by grouping and color-coding feedback items to differentiate between critical feedback, suggestions for improvement, and positive comments. A priority will be further improving the quality of the feedback provided. Integrating direct automated feedback into other courses and universities, and empirically evaluating its effects on learning experience, performance, and motivation is another crucial step. Further, we plan to consider potential issues with AI in education, such as over-reliance and ethical implications in the learning process.

A Acknowledgments

This paper includes sections revised with the help of ChatGPT to enhance clarity and readability. We have carefully reviewed all text produced by AI. The datasets for the evaluation discussed in this paper are available on OSF[9].

[9] https://osf.io/p9szv/?view_only=7d0b9889657244dcaee896070d7c15da

References

[1] Patricia A. Alexander and P. Karen Murphy. 1998. Profiling the Differences in Students' Knowledge, Interest, and Strategic Processing. *Journal of Educational Psychology* 90, 3 (1998), 435–447.

[2] I Elaine Allen and Christopher A Seaman. 2007. Likert Scales and Data Analyses. *Quality progress* 40, 7 (2007), 64–65.

[3] Xavier Amatriain. 2024. Prompt Design and Engineering: Introduction and Advanced Methods. arXiv:2401.14423 [cs]

[4] Pasquale Ardimento, Mario Luca Bernardi, and Marta Cimitile. 2024. Teaching UML Using a RAG-based LLM. In *2024 International Joint Conference on Neural Networks (IJCNN)*. 1–8.

[5] Pasquale Ardimento, Mario Luca Bernardi, Marta Cimitile, and Michele Scalera. 2024. A RAG-based Feedback Tool to Augment UML Class Diagram Learning. In *Proceedings of the ACM/IEEE 27th International Conference on Model Driven Engineering Languages and Systems (MODELS Companion '24)*. Association for Computing Machinery, 26–30.

[6] Anonymized Author. 2018. <placeholder due to the anonymized review>.

[7] Imen Azaiz, Natalie Kiesler, and Sven Strickroth. 2024. Feedback-Generation for Programming Exercises With GPT-4. arXiv:2403.04449 [cs]

[8] Brett A. Becker, Paul Denny, James Finnie-Ansley, Andrew Luxton-Reilly, James Prather, and Eddie Antonio Santos. 2023. Programming Is Hard - Or at Least It Used to Be: Educational Opportunities and Challenges of AI Code Generation. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1 (SIGCSE 2023)*. Association for Computing Machinery, 500–506.

[9] Zana Buçinca, Maja Barbara Malaya, and Krzysztof Z. Gajos. 2021. To Trust or to Think: Cognitive Forcing Functions Can Reduce Overreliance on AI in AI-assisted Decision-making. *Proc. ACM Hum.-Comput. Interact.* 5, CSCW1 (April 2021), 188:1–188:21.

[10] Javier Cámara, Javier Troya, Lola Burgueño, and Antonio Vallecillo. 2023. On the Assessment of Generative AI in Modeling Tasks: An Experience Report with ChatGPT and UML. *Software and Systems Modeling* 22, 3 (June 2023), 781–793.

[11] Phillip Dawson, Michael Henderson, Paige Mahoney, Michael Phillips, Tracii Ryan, David Boud, and Elizabeth Molloy. 2019. What Makes for Effective Feedback: Staff and Student Perspectives. *Assessment & Evaluation in Higher Education* 44, 1 (Jan. 2019), 25–36.

[12] Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, Meng Wang, and Haofen Wang. 2024. Retrieval-Augmented Generation for Large Language Models: A Survey. arXiv:2312.10997 [cs]

[13] Marcelo Guerra Hahn, Silvia Margarita Baldiris Navarro, Luis De La Fuente Valentin, and Daniel Burgos. 2021. A Systematic Review of the Effects of Automatic Scoring and Automatic Feedback in Educational Settings. *IEEE Access* 9 (2021), 108190–108198.

[14] Michael Henderson, Tracii Ryan, and Michael Phillips. 2019. The Challenges of Feedback in Higher Education. *Assessment & Evaluation in Higher Education* 44, 8 (Nov. 2019), 1237–1252.

[15] Richard Higgins, Peter Hartley, and Alan Skelton. 2002. The Conscientious Consumer: Reconsidering the Role of Assessment Feedback in Student Learning. *Studies in Higher Education* 27, 1 (Feb. 2002), 53–64.

[16] Lei Huang, Weijiang Yu, Weitao Ma, Weihong Zhong, Zhangyin Feng, Haotian Wang, Qianglong Chen, Weihua Peng, Xiaocheng Feng, Bing Qin, and Ting Liu. 2023. A Survey on Hallucination in Large Language Models: Principles, Taxonomy, Challenges, and Open Questions. arXiv:2311.05232 [cs]

[17] Alastair Irons. 2007. *Enhancing Learning through Formative Assessment and Feedback*. Routledge.

[18] Enkelejda Kasneci, Kathrin Sessler, Stefan Küchemann, Maria Bannert, Daryna Dementieva, Frank Fischer, Urs Gasser, Georg Groh, Stephan Günnemann, Eyke Hüllermeier, Stephan Krusche, Gitta Kutyniok, Tilman Michaeli, Claudia Nerdel, Jürgen Pfeffer, Oleksandra Poquet, Michael Sailer, Albrecht Schmidt, Tina Seidel, Matthias Stadler, Jochen Weller, Jochen Kuhn, and Gjergji Kasneci. 2023. ChatGPT for Good? On Opportunities and Challenges of Large Language Models for Education. *Learning and Individual Differences* 103 (April 2023), 102274.

[19] Majeed Kazemitabaar, Xinying Hou, Austin Henley, Barbara Jane Ericson, David Weintrop, and Tovi Grossman. 2024. How Novices Use LLM-based Code Generators to Solve CS1 Coding Tasks in a Self-Paced Learning Environment. In *Proceedings of the 23rd Koli Calling International Conference on Computing Education Research (Koli Calling '23)*. Association for Computing Machinery, 1–12.

[20] Hieke Keuning, Johan Jeuring, and Bastiaan Heeren. 2018. A Systematic Literature Review of Automated Feedback Generation for Programming Exercises. *ACM Transactions on Computing Education* 19, 1, Article 3 (Sept. 2018).

[21] Natalie Kiesler, Dominic Lohr, and Hieke Keuning. 2023. Exploring the Potential of Large Language Models to Generate Formative Programming Feedback. In *2023 IEEE Frontiers in Education Conference (FIE)*. 1–5.

[22] Stephan Krusche, Andreas Seitz, Jürgen Börstler, and Bernd Bruegge. 2017. Interactive Learning: Increasing Student Participation through Shorter Exercise Cycles. In *Proceedings of the Nineteenth Australasian Computing Education Conference (ACE '17)*. Association for Computing Machinery, 17–26.

[23] Juho Leinonen, Paul Denny, and Jacqueline Whalley. 2022. A Comparison of Immediate and Scheduled Feedback in Introductory Programming Projects. In *Proceedings of the 53rd ACM Technical Symposium on Computer Science Education - Volume 1 (SIGCSE 2022, Vol. 1)*. Association for Computing Machinery, 885–891.

[24] Mark Liffiton, Brad E Sheese, Jaromir Savelka, and Paul Denny. 2024. CodeHelp: Using Large Language Models with Guardrails for Scalable Support in Programming Classes. In *Proceedings of the 23rd Koli Calling International Conference on Computing Education Research (Koli Calling '23)*. Association for Computing Machinery, 1–11.

[25] Yi Liu, Gelei Deng, Zhengzi Xu, Yuekang Li, Yaowen Zheng, Ying Zhang, Lida Zhao, Tianwei Zhang, Kailong Wang, and Yang Liu. 2024. Jailbreaking ChatGPT via Prompt Engineering: An Empirical Study. arXiv:2305.13860 [cs]

[26] Samiha Marwan, Ge Gao, Susan Fisk, Thomas W. Price, and Tiffany Barnes. 2020. Adaptive Immediate Feedback Can Improve Novice Programming Engagement and Intention to Persist in Computer Science. In *Proceedings of the 2020 ACM Conference on International Computing Education Research (ICER '20)*. Association for Computing Machinery, 194–203.

[27] Ha Nguyen and Vicki 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)*. Association for Computing Machinery, 958–964.

[28] Jakob Nielsen. 2000. Why You Only Need to Test with 5 Users.

[29] Per Runeson and Martin Höst. 2009. Guidelines for Conducting and Reporting Case Study Research in Software Engineering. *Empirical Software Engineering* 14, 2 (April 2009), 131–164.

[30] Valerie J. Shute. 2008. Focus on Formative Feedback. *Review of Educational Research* 78, 1 (March 2008), 153–189.

[31] H. Sondergaard and D. Thomas. 2004. Effective Feedback to Small and Large Classes. In *34th Annual Frontiers in Education, 2004. FIE 2004.* IEEE, 540–545.

[32] Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, Ed H. Chi, Tatsunori Hashimoto, Oriol Vinyals, Percy Liang, Jeff Dean, and William Fedus. 2022. Emergent Abilities of Large Language Models. *Transactions on Machine Learning Research* (June 2022).

[33] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc V. Le, and Denny Zhou. 2022. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. *Advances in Neural Information Processing Systems* 35 (Dec. 2022), 24824–24837.

[34] Roel J. Wieringa. 2014. *Design Science Methodology for Information Systems and Software Engineering*. Springer Berlin Heidelberg.

[35] Juliette Woodrow, Ali Malik, and Chris Piech. 2024. AI Teaches the Art of Elegant Coding: Timely, Fair, and Helpful Style Feedback in a Global Course. In *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1 (SIGCSE 2024)*. Association for Computing Machinery, 1442–1448.

[36] Jingfeng Yang, Hongye Jin, Ruixiang Tang, Xiaotian Han, Qizhang Feng, Haoming Jiang, Shaochen Zhong, Bing Yin, and Xia Hu. 2024. Harnessing the Power of LLMs in Practice: A Survey on ChatGPT and Beyond. *ACM Trans. Knowl. Discov. Data* 18, 6 (April 2024), 160:1–160:32.

[37] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. 2023. ReAct: Synergizing Reasoning and Acting in Language Models. In *The Eleventh International Conference on Learning Representations*.

[38] Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, Yifan Du, Chen Yang, Yushuo Chen, Zhipeng Chen, Jinhao Jiang, Ruiyang Ren, Yifan Li, Xinyu Tang, Zikang Liu, Peiyu Liu, Jian-Yun Nie, and Ji-Rong Wen. 2023. A Survey of Large Language Models. arXiv:2303.18223 [cs]