Adaptive Learning Systems in Programming Education: A Prototype for Enhanced Formative Feedback

Dominic Lohr [1], Marc Berges [1], Abhishek Chugh[2], and Michael Striewe [3]

Abstract:

Formative feedback is crucial in programming education, yet many learning systems fall short, concentrating mostly on pinpointing errors rather than guiding learners on how to resolve them. This is particularly unhelpful for novices who often lack advanced skills like debugging. Feedback is considered more valuable when it addresses error causes rather than just symptoms. However, this is challenging using only conventional methods like unit testing. Identifying error causes requires detailed information about both the error and the learner. Our proposed prototype introduces a new approach to integrating programming exercises into adaptive learning systems. It directly categorizes student code into so-called *answer classes* using a combination of static and dynamic code analysis. When integrated with data derived from a learner model, this approach enables tailored feedback that lowers the barrier to learning programming while keeping motivation high.

Keywords: programming education, adaptive feedback, CS1

1 Motivation

Programming educators in higher education are faced with the challenge of teaching diverse groups of learners whose prior knowledge of programming varies significantly [Ca15]. This discrepancy often results from the different school systems and curricula that students have passed through [Hu15]. While some learners have never written a single line of code others already have experience in developing (small) projects. Not overwhelming the former and not boring the latter is a pedagogical balancing act for educators, especially in the first few sessions of a programming course.

Adaptive learning systems (ALS) are considered as a promising approach to meet the heterogeneous needs of learners. Especially for programming novices, effective feedback is seen as crucial to support learners. However, the type of feedback required varies greatly depending on the individual learning level and type of error. Although providing elaborated feedback is crucial for learners, most learning systems offer only simple feedback focused mainly on mistakes [KJH19, Je22], without customization to individual needs. This often stems from the challenge of linking mistakes to their underlying causes, as many systems

1 FAU, Computer Science Education, Martensstraße 3, 91058 Erlangen, Germany,
dominic.lohr@fau.de, https://orcid.org/0000-0002-6330-2327;
marc.berges@fau.de, https://orcid.org/0000-0002-9982-547X
2 https://sophize.org, abc@sophice.de
3 University of Duisburg-Essen, The Ruhr Institute for Software Technology, Gerlingstraße 16, 45127 Essen, Germany, michael.striewe@paluno.uni-due.de, https://orcid.org/0000-0001-8866-6971

doi:10.18420/delfi2024_57

lack a model of the learner's unique profile, which we claim is crucial in most cases to infer from symptoms to causes – considered as more valuable feedback by learners [LB21]. The presented demo showcases a prototype that illustrates how an adaptive learning system can address these challenges. It achieves this by integrating diagnostic features with adaptive learning functionalities, merging established systems with a large language model (LLM).

2 Architecture

The architecture is composed of three integrated components: (A) the JACK system, (B) the ALEA system, and (C) GPT4 – a state-of-the-art large language model.

To evaluate the learner's program code we use JACK – an e-assessment system designed for grading programming exercises [St16]. JACK allows for both static and dynamic analysis of Java code. Based on these analyses we map the program code to specific answer classes. From a large number of existing systems for automated feedback generation for programming exercises [SS22], JACK is a suitable choice in this endeavor for two reasons: First, the architecture of JACK is specifically designed for modularity and extensibility, what makes it easy to integrate only the required components from JACK into the new prototype. Second, JACK employs a dedicated, customizable module for static code analysis [SG14], that allows the creation of exercise-specific rules and thus produces more fine-grained answer classes.

ALEA is an adaptive learning assistant for university courses that can provide customized content for learners based on semantically annotated course materials [Kr23, Be23]. In ALEA, learning objects are annotated following the Y-model framework [Lo23]. This approach makes details regarding necessary prerequisite knowledge, relevant concepts, learning objectives, and answer classes available to the system. Additionally, ALEA incorporates a learner model, capable of offering insights into the concepts that learners have successfully grasped within a specific cognitive dimension.

We use GPT4 to generate the feedback messages as the potential of the model to generate feedback has already been demonstrated in studies [KLK23, LKK24]. However, research shows that the generated feedback often lacks customization and, therefore, adaptivity. By applying retrieval augmented generation (RAG), we can dynamically add specific information to the prompt, making the feedback more tailored to the learner's needs.

2.1 System Design

Fig. 1 gives an overview of the architecture and the feedback pipeline. When feedback is requested, the code is first sent to the JACK to analyze whether it is compilable code. If the code contains syntax errors, there are three possibilities: (1) the compiler error message is sent directly to the learner (2) an explanation of the error message is generated with GPT4 and sent to the learner or (3) the program code is repaired to compilable code using GPT4

and then examined using the JACK system with regard to the answer classes. The latter variant also enables the provision of feedback on semantic or logical errors if the program code does not compile. This fundamentally expands the possibilities, as previous tools were unable to provide feedback on logical or semantic errors in non-compilable code.

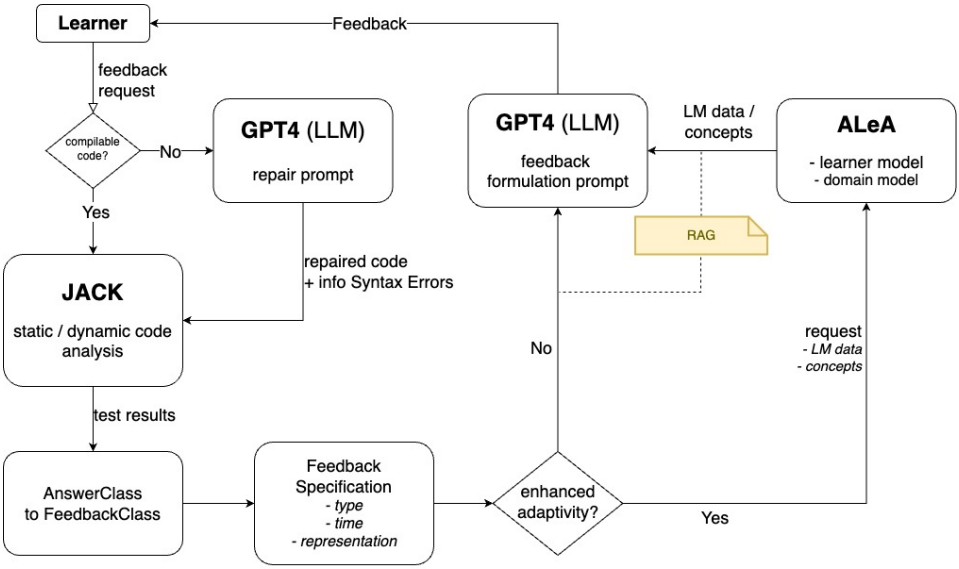

Fig. 1: Overview of system design

2.2 Functionality

Fig. 2 shows a screenshot of the prototype[4]. While working on a task, learners can use various methods to request different types of feedback, and it is up to the learner to decide when and how to ask for help. This fosters self-directed learning, as learners have different preferences. We also don't want to interrupt learners during their cognitive processes (a 'thinking step'). Therefore, we leave the timing of the feedback to the learner but implement automated support functionality when obviously struggling with the task, e. g. trying to compile five times with the same syntax error. Feedback from the compiler can be requested by using the RUN button. If there are test cases provided in the task the learner can request the test results using the TEST button. When struggling with the task the learner can request help any time using the HELP functionality.

4 Example task and code is taken from the Coding-Bat dataset [Ki22]

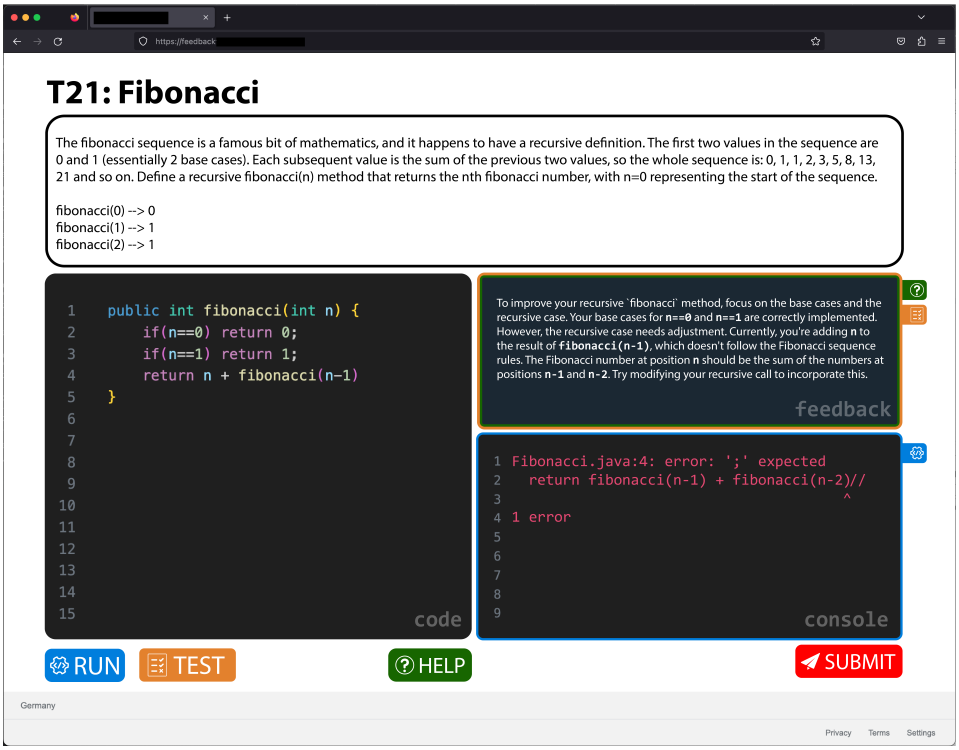

Fig. 2: Screenshot of the Prototype

3 Future Work and Vision

The prototype presented in this demo is limited to the programming language Java. One reason for this is that the JACK system currently only supports an analysis of Python and Java tasks. However, we claim that the system presented generalizes well to other programming languages – presumably with a few adaptations. Developing suitable answer classes for the programming tasks and writing suitable tests requires a considerable amount of time for programming instructors. However, some of the answer classes can be generalized across task types. It is, therefore, conceivable that in the future, the educator already has a well-stocked toolbox of answer classes with the appropriate analysis tests available for certain programming tasks.

Automated adaptive programming feedback offers a solution to the diverse needs of students in large introductory courses. Traditionally, the lack of time and resources makes it challenging for educators to provide personalized support, especially to those who need it most. Our approach demonstrates an initial step towards providing automatic, tailored feedback, addressing this critical gap.

Bibliography

[Be23] Berges, Marc; Betzendahl, Jonas; Chugh, Abhishek; Kohlhase, Michael; Lohr, Dominic; Müller, Dennis: Learning Support Systems Based on Mathematical Knowledge Management. In (Dubois, Catherine; Kerber, Manfred, eds): Intelligent Computer Mathematics, volume 14101, pp. 84–97. Springer Nature Switzerland, Cham, 2023.

[Ca15] Capovilla, Dino; Berges, Marc; Mühling, Andreas; Hubwieser, Peter: Handling Heterogeneity in Programming Courses for Freshmen. In: 3rd International Conference on Learning and Teaching in Computing and Engineering (LaTiCE). IEEE Press, Los Alamitos, pp. 197–203, 2015.

[Hu15] Hubwieser, Peter; Giannakos, Michail N.; Berges, Marc; Brinda, Torsten; Diethelm, Ira; Magenheim, Johannes; Pal, Yogendra; Jackova, Jana; Jasute, Egle: A Global Snapshot of Computer Science Education in K-12 Schools. In (Ragonis, Noa; Kinnunen, Päivi, eds): Proceedings of the 2015 ITiCSE on Working Group Reports. ACM Press, New York, pp. 65–83, 2015.

[Je22] Jeuring, Johan; Keuning, Hieke; Marwan, Samiha; Bouvier, Dennis; Izu, Cruz; Kiesler, Natalie; Lehtinen, Teemu; Lohr, Dominic; Peterson, Andrew; Sarsa, Sami: Towards Giving Timely Formative Feedback and Hints to Novice Programmers. In: Proceedings of the 2022 Working Group Reports on Innovation and Technology in Computer Science Education. ACM, Dublin Ireland, pp. 95–115, December 2022.

[Ki22] Kiesler, Natalie: Dataset: Recursive problem solving in the online learning environment CodingBat by computer science students. Online, June 2022. Datenerhebung: 2017. Version: 1.0.0. Datenpaketzugangsweg: Download-SUF. Hannover: FDZ-DZHW. Datenkuratierung: İkiz-Akıncı, Dilek.

[KJH19] Keuning, Hieke; Jeuring, Johan; Heeren, Bastiaan: A Systematic Literature Review of Automated Feedback Generation for Programming Exercises. ACM Transactions on Computing Education, 19(1):1–43, March 2019.

[KLK23] Kiesler, Natalie; Lohr, Dominic; Keuning, Hieke: Exploring the Potential of Large Language Models to Generate Formative Programming Feedback. In: 2023 IEEE Frontiers in Education Conference (FIE). IEEE, College Station, TX, USA, pp. 1–5, October 2023.

[Kr23] Kruse, Theresa; Berges, Marc; Betzendahl, Jonas; Kohlhase, Michael; Lohr, Dominic; Müller, Dennis: Learning with ALeA: Tailored Experiences through Annotated Course Material. In: INFORMATIK 2023 - Designing Futures: Zukünfte Gestalten. Gesellschaft für Informatik e.V., Bonn, pp. 395–398, 2023.

[LB21] Lohr, Dominic; Berges, Marc: Towards Criteria for Valuable Automatic Feedback in Large Programming Classes. In: Hochschuldidaktik Informatik HDI 2021. Jörg Desel, Simone Opel, Dortmund, pp. 181–186, September 2021.

[LKK24] Lohr, Dominic; Keuning, Hieke; Kiesler, Natalie: You're (Not) My Type – Can LLMs Generate Feedback of Specific Types for Introductory Programming Tasks? Journal of Computer Assisted Learning, 2024. accepted.

[Lo23] Lohr, Dominic; Berges, Marc; Kohlhase, Michael; Müller, Dennis; Rapp, Max: The Y-Model - Formalization of Computer Science Tasks in the Context of Adaptive Learning Systems. In: 2023 IEEE 2nd German Education Conference (GECon). IEEE, Berlin, Germany, pp. 1–6, August 2023.

[SG14] Striewe, Michael; Goedicke, Michael: A Review of Static Analysis Approaches for Programming Exercises. In: Proceedings of the International Conference on Computer Assisted Assessment (CAA 2014). CCIS 439, Zeist, Netherlands, pp. 100–113, 2014.

[SS22] Strickroth, Sven; Striewe, Michael: Building a Corpus of Task-Based Grading and Feedback Systems for Learning and Teaching Programming. International Journal of Engineering Pedagogy (iJEP), 12(5):26–41, Nov. 2022.

[St16] Striewe, Michael: An Architecture for Modular Grading and Feedback Generation for Complex Exercises. Science of Computer Programming, 129:35–47, November 2016.