UNIVERSITY OF CALIFORNIA

SANTA CRUZ

A GENERALIZABLE CONSTRAINT-BASED EXPLANATION SYSTEM FOR EDUCATION IN COMPUTER SCIENCE

A project submitted in partial satisfaction

of the requirements for the degree of

MASTER OF SCIENCE

in

COMPUTER SCIENCE AND ENGINEERING

by

Aaja Ouellette

June 2025

The Master's Project of
Aaja Ouellette is approved:

DocuSigned by:
Leilani Gilpin
A026316FFDEE48A...
Professor Leilani Gilpin

DocuSigned by:
Razvan Marinescu
A446AE995AA545A...
Professor Razvan Marinescu

Peter Biehl
Vice Provost and Dean of Graduate Studies

A Generalizable Constraint-Based Explanation System for Education in Computer Science

Aaja Ouellette
aaouelle@ucsc.edu
University of California Santa Cruz
Santa Cruz, California, USA

ABSTRACT

This report relates completed and ongoing efforts towards gamification and generalizable explanation virtual tutor-like tools in Computer Science education. How does the use of an autograder augmented with the capability to offer Computer Science students immediate and automatic explanation impact the performance and experience of undergraduate students when solving programming problems similar to the type found in first and second year computer science classes? We developed and ran a two-group pilot study where intermediate-level Computer Science undergraduate students volunteered from our lab to solve a problem either with or without the assistance of our explanation system. The results of the two groups were compared to ascertain what effects, if any, the tool had on student experience and performance, with results showing no conclusive benefit nor penalty attributable to the tool's presence. However, there may have been confounding factors, such as the low number of participants and the participants being more experienced than those the exercise was initially designed for, that contributed to this result. Additionally, qualitative feedback was collected from participants to help guide the future development of this tool.

ACM Reference Format:
Aaja Ouellette. 2025. A Generalizable Constraint-Based Explanation System for Education in Computer Science. In *NotARealJournal 25, Santa Cruz, CA.* ACM, New York, NY, USA, 6 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

Computer science is one of the fastest-growing disciplines in academia, with increasingly large classes that can easily rise into the hundreds. These class sizes, and indeed even classes with sizes far smaller, are beyond the means of teaching staff to provide personalized and timely feedback to each student—attention from which they undoubtedly benefit from. That this difference is being made up with automated grading systems that have often no ability, or only quite a limited one, to provide any kind of feedback at all to students beyond a grade.

In this report, we detail the system we have made to respond to this issue. Taking an approach inspired by intelligent tutoring systems (ITS), which have been demonstrated to be an effective tool at improving student learning gains. we have worked to create a generalizable tool that can interpret student code and provide immediate feedback to the student as they work. This approach is inspired by the effectiveness of ITS, which have been shown to improve student learning gains (Pane et al., 2013). In particular, our approach is to apply constraint-based modeling (CBM), which is a form of ITS that identifies specific traits that every solution must satisfy, and then guides the learner towards fulfilling those traits, to the completion of programming assignments (Ohlsson, 2016).

After developing the initial design and prototype of our system, we sought to evaluate it over the course of a pilot study. This pilot study was formulated to discover the performance and experience of beginner Computer Science students as they worked on a programming exercise. Participants were divided into two groups, with participants in the first group working on the exercise while supported by the explanation system we developed, while participants in the second group attempted the exercise unsupported. The progress participants made on the programming exercise after thirty minutes, as well as their answers to post-test survey composed of a number of Likert scale quantitative questions and an additional set of qualitative questions for the participants in the group working with the tool, and the differences between the two groups, inform our results and conclusions about the effectiveness of the tool.

2 RELATED WORK

Intelligent tutoring systems are certainly not a new idea, but they are usually used in only a few traditional ways (Crompton & Burke, 2023). Furthermore, many of the most common and prominent designs behind ITS are not wholesale convertible for the type of assignment common in Computer Science. For example, cognitive tutors function by assessing and reassessing subject knowledge and then selecting material to present to students (Steenbergen-Hu, S., & Cooper, H. (2014). This does not match traditional method of teaching Computer Science classes and assignments, which focuses on the student working their way through a large assignment from beginning to end and the implementation of algorithms, leaning more towards project-based work and with little emphasis on the back-and-forth type of learning that a cognitive tutor is designed for. Other times, AI personalized feedback resembles notes or guidance given to students particular to the background and standing in the class, and is given over the duration of an entire course rather than on a step-by-step basis over the course of a single assignment, such as the Course Signals tool at Purdue University and the Expert Electronic Coach at the University of Michigan, as well as many others (Mousavi, 2021). This sort of feedback, although proven to

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-1-4503-XXXX-X/18/06
https://doi.org/XXXXXXX.XXXXXXX

be useful, does not necessarily do much to help the student at a low, granular level. When feedback by intelligent tutoring systems is instead given with respect to only a single exercise, this is a laborious task that requires both expert knowledge and significant time to write out detailed explanations for each step, and each mistake that can be made in this step, before it is properly useful to the student.

We know that the granularity, that is, how large the reasoning of each step made between feedback is in a given context, of an intelligent tutoring system is highly significant when it comes to how much that system is able to support and improve the scores of students. For example, a system that can give feedback to a single incorrect step in a longer solution has low granularity and has been shown to significantly outperform systems with larger granularity, such as one that can only provide feedback to the entire solution as a whole (VanLehn, 2011). In other words, having low granularity is an important quality for an effective intelligent tutoring system, but it also necessarily increases the complexity of the system. A more complex system can give more complex feedback to students, at the cost of a more complex design and implementation.

We are also far from the only group to believe abstract syntax trees hold potential for delivering automatic student feedback. Other work investigates the ability of an automated, explainable AST-based approach to detecting logical errors made by students and predicting their performance in programming courses (Hoq et al., 2025). This work, however, leaves the next step of delivering immediate, automatic, and personalized to students as future work. Being that our approach is purely rule-based, our simpler modeling of the problem allows us to present feedback more easily to students. Further research into the problem will reveal the relative strengths and weaknesses between these two different approaches.

The motivation behind the project detailed in this report is, then, to make progress towards developing an intelligent tutoring system that retains the ability to give complex and low-granularity feedback to students, while minimizing the amount of time and effort that is necessary for its proper implementation. In other words, we have been working to develop a generalizable explanation system that can be tuned to provide low-granularity explanations for different problems, all while retaining the same infrastructure. With comparatively lower effort, though necessitating expert knowledge to properly reorient it, the system can be repurposed to provide explanations to different programming problems. This ITS can isolate and identify differences between submitted student code and staff-provided solutions using abstract syntax trees and assist students in honing in on the areas where the issues reside. Once located, it can provide low granularity, step-based feedback to the user.

3 METHODS

3.1 Study Design

The pilot study was created to assess the effectiveness of the explanation tool on both student progress on the problem during the session and on student experience. We decided to design the programming exercise used in the study for first-year and second-year students, as this would allow the implementation of both the problem and explanation system to be simpler, given that students earlier on in their educations are naturally working on simpler problems than those later on in their educations. The programming problem was based on an equivalent-level problem from ECS 36A: Programming and Problem Solving, a course taught at UC Davis (Bishop, 2024). The problem was designed to be solvable for participants without too much difficulty, limited to topics such as basic single-nested looping, printing, and string manipulation, so participants would be able to start with no need for additional instruction. Participants were expected to be able to solve simple cases without too much effort, but there were also several hidden edge cases present that were likely to cause errors. Additionally, the grading function would stop after a test case had failed and the system's explanation would be output. This all was to create additional opportunities for the student to engage with the explanations, as they were very likely to have to submit at least several times, and each time were faced with the system's explanation.

Quantitative Questions For Both Groups	Competency
I am comfortable programming in Python	Reading and understanding code
I am comfortable using loops in Python	Reading and understanding code
I am comfortable using string manipulation in Python	Reading and understanding code
I found it difficult to actually begin the problem	Understanding a problem
I found it difficult to understand what the problem was asking me to do	Understanding a problem
It was difficult to turn my ideas for solving the problem into code	Understanding a problem
I had a good idea of the structure I wanted for my solution	Understanding a problem
I got stuck at some points while writing my solution	Debugging
I found it easy to debug my solution	Debugging
I found it easy to expand my solution to cover all test cases	Debugging
I was frustrated when solving some parts of the problem	Student experience
I felt lost when solving some parts of the problem	Student experience

Figure 1: Quantitative questions asked during the pilot study. Participants were presented with a Likert scale from 1 to 5 for all questions.

Student performance was gathered each time the student submitted, as the exercise would log the timestamp of the submission as well as which of the six test cases the student's submission had passed. This was to identify if students supported by the explanation system were able to make more consistent progress without large gaps between submissions thanks to the tool's guidance when compared to their counterparts in the other group. This metric can also be used to indicate some level of student engagement based on time between submissions. Finally, these logs would also allow us to see the time between the very first and last submissions, measuring how long the student worked on the assignment, and the number of test cases they had successfully passed by the end.

Student experience, on the other hand, was collected immediately after the exercise was completed through a post-test questionnaire. This questionnaire invited participants to rate their own aptitude and then answer questions about their experience during the exercise itself. In particular, participants were asked to evaluate their performance with regards to different skills, modeled after the breakdown used by Hammer to measure programming competencies (Hammer, 2018). Some competencies were modified or dropped

Qualitative Questions for Explanation Group
Do you think the explanation tool improved your experience solving this problem?
Would you like it if this tool was a part of solving assignments in your classes?
What aspects of this experience need improvement and what changes would you suggest?
Please provide any additional comments on this experience here

Figure 2: Qualitative questions asked during the pilot study. Participants were presented with a short answer text-box for all questions.

to more closely match the problem selected for this pilot study. For example, the competence 'programming at the machine level' was dropped because it does not apply to the exercise conducted during this pilot study. In order to prevent participant survey overload, the number of questions was kept low.

In addition, participants in the group supported by the explanation system were given an additional set of qualitative questions asking for feedback about their experience with the system and the experience as a whole. The qualitative questions were adapted to target the student's affective experience of the survey (Barrie et al., 2015).

3.2 Explanation System

As for the explanation system itself, the systems receives source code each time the participant submits the assignment, as explained above. From there, the code is parsed as an abstract syntax tree. The system is not intended to to debug participant code on their behalf, rather, it is meant to identify the area where the student's submitted code differs from expectation and bring it to their attention. The main idea behind the system is to help the student avoid becoming stuck while working and the explanation system aims to address this problem by giving the student an idea about what is going wrong.

On a high level, the system makes use of what we have called 'structural explanations' to produce feedback. This means the system looks at the structure of the student's code and compares it to an expected structure. The system knows where there are differences and brings these to the student's attention. As it stands now, the expected structure has to be specified by hand and by someone possessing expert knowledge of the exercise. They must decide what elements are important and crucial to the structure of a solution, and which are not. In addition to the node matching or lack thereof produced by the explanation system, the designer is also able to add in any particular phrases in the feedback that they might wish to include.

In addition to matching against the nodes in the AST produced by the student's code, it is also possible to try to match against specific strings present therein. This could be desirable if, for instance, the student is intended to use a certain helper function to make the problem much easier than it might otherwise be. In that case, the designer could add a rule in that checks to see if the student uses the function in the intended place and remind them if they do not.

Within the inner workings of the explanation system, the student's submitted source code is processed as an abstract syntax tree, or AST, which allow us to search over the student's code in a generalizable way, not limited by some conventions or different naming schemes. Once this is done, we begin matching each ruleset against the student's AST, one at a time. We make a special distinction for the nodes that are at the highest level. In programming terms, these would be all the terms that are at the highest level of indentation within the source code, not subordinated to anything else than the function definition itself. We call these 'top level nodes.' When it comes to matching each ruleset, we try to match the first rule against one of the top level nodes. Once we've found a match, enter the sub-tree which has the top-level node as a root to try to match the rest of the ruleset.

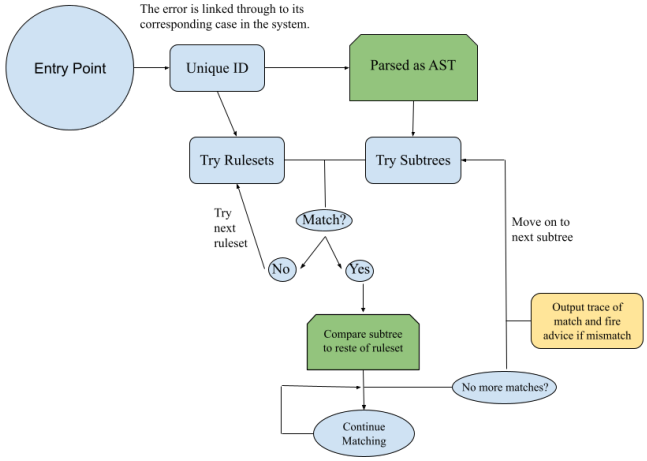

Figure 3: High-level diagram of the explanation system. Participant code is parsed as an AST and analysed by matching pre-defined rules against its structure.

Within the subtree, we try to match rules against the AST one at a time, in sequence. Some rules are given as datatypes, like *ast.Return* corresponding to a *return* statement. Other rules are expressed as strings, which unparse some part of the subtree in question and can check the code directly as a string. The reason you might want to do this is to check to see if the student is using a certain function, for example. That is, if a student is intended to use a helper function that makes the problem easier, this is a way to try to remind them to do so. The matching continues until either a rule cannot be matched within the local context or all the rules have been successfully matched. Once completed, we repeat the process from the top level node step with the next ruleset.

3.3 Procedure

The pilot study was conducted by a small group of lab members with participants made up of undergraduate students of intermediate experience from the lab. The study began with a very brief presentation about the pilot study and what we were trying to achieve with it, before dividing the participants into two groups, the group supported by the explanation system and the group not supported. Afterwards, all participants were given 30 minutes to complete the exercise. Finally, participants completed the survey detailed above before listening to a short debriefing.

4 RESULTS

In terms of quantitative data, neither of the results of both the group supported by the explanations (left in the figure below) and the group not supported by the explanations (right) showed much significant variation. If anything, the group that was supposed to benefit from the explanations was instead impeded by them, reporting lower average scores to several positive questions (e.g. 'I am comfortable programming in Python') and higher average scores to negative questions (e.g. 'I was frustrated when solving some parts of the problem.')

The qualitative data we obtained during the pilot study tells a similarly mixed story. One participant had a slightly favourable view of the system "It did help when it was available, but whenever my program didn't fully run there was no helpful output," but noted that, when their program crashed, the explanations were not run: "There is no helpful output when there program has an error and doesn't run to completion. I don't know if this is something you can do, but if you could find a way to detect the error that occurs and not have the program crash offer suggestions about why the program crashed." This is a good point, and certainly something that could be added to the system. Another participant, despite the briefing at the start of the session, did not notice the explanation system at all: "I did not realize there was one. I was trying to debug it myself." This is surprising, as the explanations were included directly below the output from the testing script, but it does emphasize the importance of presentation. A third participant found the output confusing, particularly the way that the system expressed what nodes in the participant's AST were matched and which were not. The participant found the tool to be confusing and recorded that, as the system acted in a way they were not expecting it to, they assumed it was an error.

Responding to the next question, the majority of participants reported that they would be in favour of this tool being a part of solving assignments in their classes, with the caveat that it was a bit more polished and the output was more clear, which were also the biggest suggestions or changes that participants suggested for the tool in the final question. The rest of the responses were indifferent to the tool's adoption or not, and there were not any negative responses.

5 DISCUSSION

Some important notes, however, should be considered when looking at the results. First, the number of participants was quite low, totaling only 11, with five in the explanations group and six in the other. This is far below the minimum number necessary to be statistically significant, by the Central Limit Theorem. As such, any and all results can be called into question. Next, the participants were more experienced than originally intended for the study, which may have been a confounding factor and influenced results, either positively or negatively. As previously related, the exercise's difficulty was intended for Computer Science students early in their education and without much experience. Due to significant, repeated difficulties in gathering appropriate candidates and, despite our best efforts, two false starts, we eventually opted to conduct the pilot study with lab members as participants. As a result, only one of the participants was at the intended level, with one more being one course ahead, and the rest being at least two entire courses ahead of expectations. It could be that the experience of the participants meant that they were able to complete the exercise with ease and did not benefit from the explanations because the problem was not sufficiently challenging for them to require it. Due to these factors, conclusive evidence regarding the effects of this system on student outcomes and experience has yet to be produced.

In any case, however, the qualitative responses have shown clear examples of ways to improve the explanation system. To begin with, it's undoubtedly true that the system could use more polish, clarity, and better representation in the output of the feedback. However, this is not a trivial problem, as feedback is being returned alongside grading results through the terminal. Needless to say, any feedback presented through the terminal would benefit greatly from any clarity that could be provided to it. Additional work would be necessary to devise and test a system that could more effectively achieve this. It is possible, however, that the nature of the way students interact with the autograder tool that the explanation system is designed to function inside of may soon change. The developers of the autograder are currently working on a web-interactive version of the system. This opens up many new avenues for formatting and delivering feedback. The adoption of this web-based UI may contribute greatly to enhanced clarity and presentation of feedback, and future development of our tool will continue to pursue this end.

The first participant's suggestion to extend explanations to cases where the system crashes could also be done. It is worth considering a special case firing specifically during crashes. Since the explanations should be specified by someone with knowledge, or at least access to knowledge, of the problem, explanations could focus on common reasons for crashes related to the particular exercise. Lists of common bug and crashes, or similar resources, often exist for the benefit of teaching staff associated with a given course. This would be an application that could broaden its reach, giving students access to common issues without needing to wait for scheduled office hours or e-mail.

There is no doubt that there remains more work to be done here, and this lab will continue this project even without the presence of this author. Once the changes suggested by participants of this study have been implemented at some point in the future, a more widely-reaching study can be conducted, this time taking additional care to achieve alignment between the difficulty of the exercise and its recipients.

In addition, future plans for development for this tool include an ability to set itself up automatically, or at least to assist the expert with that task, by scanning the solution to a given exercise and

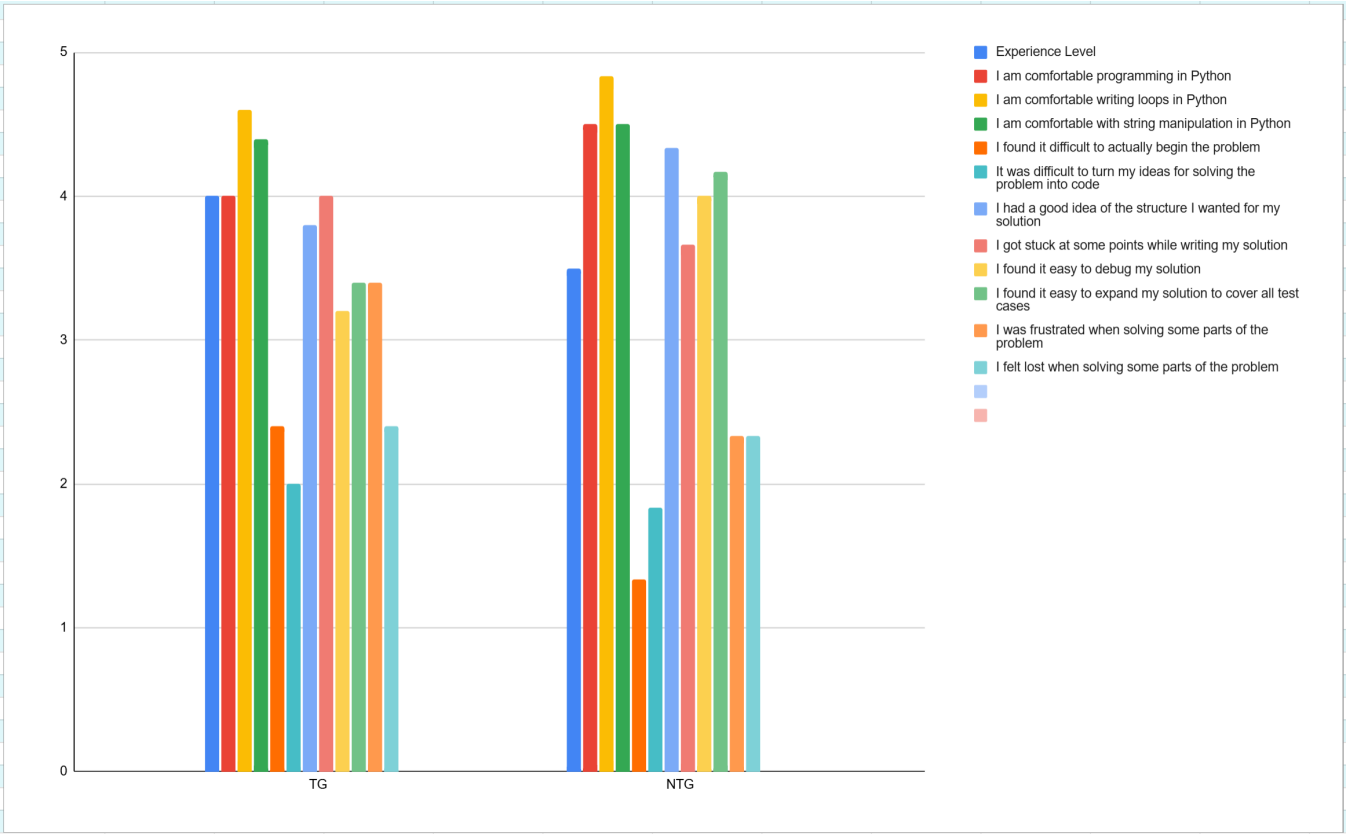

Figure 4: Average results of quantitative questions from the explanation group (left) and the unsupported group (right).

deducing rulesets from it. Such a feature would allow the tool to be more easily adoptable.

6 CONCLUSION

We developed an constraint-based explanation system to provide guidance and immediate feedback to Computer Science students working programming exercises. We conducted a pilot study to evaluate the effectiveness of our tool and found that, in its current state, it does not achieve its objectives. Nonetheless, we remain optimistic as to the tool's future based on clear qualitative feedback obtained from participants during the study, offering ideas as to how to improve the tool and stating that they would be in favour of the tool's wide adoption, provided it were a bit more polished. We will continue to develop the tool towards these ends and look forward to seeing more positive results from a study conducted some time in the future.

7 OTHER WORK

Although not directly related to the pilot study, there are also some other projects that we worked on during our time with the lab the past two years. They will be briefly mentioned here, for the record.

We replaced two assignments with new ones for CSE 240, a graduate-level artificial intelligence class here at UCSC, and added autograder support to another pre-existing assignment, which originally needed to be graded by hand. These were the first steps in an effort to unify the course's assignments to all be on a single subject. This is still an ongoing project, but after the first time the course was taught with the new assignments, where students performed very well, we have also been working on updating them to respond to student feedback collected. The first assignment was updated to be more clear with additional examples. The second assignment, which focused on implementing minimax and expectimax algorithms in chess, as well as the development of an evaluation function, has been reworked to address student concerns.

Namely, as students were developing both an algorithm and the evaluation function on which it depended, they were unsure which function was responsible for bugs during development. To address this feedback, the assignment is being refit to first grade the minimax problem using a hidden evaluation function, out of reach of the student. Then, after confirming the algorithm works properly, students will know that further issues are instead in the evaluation function. In addition, some students reported feeling daunted by needing to implement an evaluation function from scratch in the original assignment. To address this, the reworked assignment has several scaffolding puzzles which can help to guide the student as they construct their evaluation function. Rather than having to build an evaluation function that can work right out of the gate, they get the chance to apply it to small-scale scenarios designed to

encourage certain behaviour. For example, the very first puzzle may be solved simply by having your evaluation function place value on achieving checkmate. A later puzzle, however, cannot be solved in the same way, as checkmate is too far down the tree to reach. However, the student's agent can still capture certain pieces of the opponent, thus encouraging them to also make their evaluation function value capturing material. These changes should make the experience of going through the assignment have less friction, as the student will be guided through learning skills in smaller-sized pieces.

Lastly, we have also developed a new set of auditing tasks for our team within the lab. The previous tasks here were out of date and did not reflect the current state of the team's work. Although more work on the new auditing tasks remains to be done by other members of the team who work on systems that are outside of the scope of our own work, we have created a smooth introduction to the basic systems of the autograder that all projects on our team is built on, involving explanation and exercise for auditors to do. We have done the same for tasks that relate to the explanation system developed in our own work. These onboarding tasks are important to the continued operation of the team's research and will perhaps be our most impactful contribution here.

REFERENCES

[1] Barrie, S. C., Bucat, R. B., Buntine, M. A., Burke da Silva, K., Crisp, G. T., George, A. V., Jamie, I. M., Kable, S. H., Lim, K. F., Pyke, S. M., Read, J. R., Sharma, M. D., & Yeung, A. (2015). Development, Evaluation and Use of a Student Experience Survey in Undergraduate Science Laboratories: The Advancing Science by Enhancing Learning in the Laboratory Student Laboratory Learning Experience Survey. International Journal of Science Education, 37(11), 1795–1814. https://doi.org/10.1080/09500693.2015.1052585

[2] Crompton, H., & Burke, D. (2023). Artificial intelligence in higher education: the state of the field. International Journal of Educational Technology in Higher Education, 20(1), 22–22. https://doi.org/10.1186/s41239-023-00392-8

[3] VanLehn, K. (2011). The Relative Effectiveness of Human Tutoring, Intelligent Tutoring Systems, and Other Tutoring Systems. Educational Psychologist, 46(4), 197–221. https://doi.org/10.1080/00461520.2011.611369

[4] Mousavi, A., Schmidt, M., Squires, V., & Wilson, K. (2021). Assessing the Effectiveness of Student Advice Recommender Agent (SARA): the Case of Automated Personalized Feedback. International Journal of Artificial Intelligence in Education, 31(3), 603–621. https://doi.org/10.1007/s40593-020-00210-6

[5] Matt Bishop, UC Davis, https://nob.cs.ucdavis.edu/classes/ecs036a-2024-02/index.html (Last accessed November 2024)

[6] Ohlsson, S. (2016). Constraint-Based Modeling: From Cognitive Theory to Computer Tutoring – and Back Again. International Journal of Artificial Intelligence in Education, 26(1), 457–473. https://doi.org/10.1007/s40593-015-0075-7

[7] Pane, J. F., Griffin, B. A., McCaffrey, D. F., & Karam, R. (2014). Effectiveness of Cognitive Tutor Algebra I at Scale. Educational Evaluation and Policy Analysis, 36(2), 127–144. https://doi.org/10.3102/0162373713507480

[8] S. Hammer, M. Hobelsberger and G. Braun, "Using laboratory examination to assess computer programming competences: Questionnaire-based evaluation of the impact on students," 2018 IEEE Global Engineering Education Conference (EDUCON), Santa Cruz de Tenerife, Spain, 2018, pp. 355-363, doi: 10.1109/EDUCON.2018.8363251. keywords: Programming profession;Education;Tools;Task analysis;Mathematics;Laboratory Examination;Summative Assessment;Computer Programming Competences;Introductory Computer Science Course;Questionnaire-based survey assessing the impact of lab exams,

[9] Steenbergen-Hu, S., & Cooper, H. (2014). A Meta-Analysis of the Effectiveness of Intelligent Tutoring Systems on College Students' Academic Learning. Journal of Educational Psychology, 106(2), 331–347. https://doi.org/10.1037/a0034752

[10] Hoq, M., Rao, A., Jaishankar, R., Piryani, K., Janapati, N., Vandenberg, J., Mott, B., Norouzi, N., Lester, J., & Akram, B. (2025). Automated Identification of Logical Errors in Programs: Advancing Scalable Analysis of Student Misconceptions. https://doi.org/10.48550/arxiv.2505.10913