# Can Automated Feedback Turn Students into Happy Prologians?

Ricardo Brancas
INESC-ID / IST,
Universidade de Lisboa,
Portugal

Pedro Orvalho
Artificial Intelligence Research Institute,
Consejo Superior de Investigaciones Científicas,
Barcelona, Catalonia, Spain

Carolina Carreira
Carnegie Mellon University,
Pittsburgh, USA
INESC-ID / IST, Universidade de Lisboa, Portugal

Vasco Manquinho
INESC-ID / IST,
Universidade de Lisboa,
Portugal

Ruben Martins
Carnegie Mellon University,
Pittsburgh, USA

Providing personalized feedback is essential for effective learning, but delivering it promptly can be challenging in large-scale courses. In this work, we present PROHELP, an automated assessment platform for Prolog built on top of the GITSEED framework, and we evaluate it through a survey of 144 students from a 365-student undergraduate logic programming course. We assessed the perceived usefulness of seven types of automated feedback, including automatic testing, predicate scoring, syntax error highlighting, open choice point warnings, score rankings, solution type validation, and unknown predicate name suggestions. Our results show that 74% of students agreed the feedback helped increase their grade, and the system achieved a System Usability Scale score of 78.5 (grade B+). Among the feedback types, automatic testing was ranked as the most useful, followed by open choice point warnings and predicate scoring, with statistically significant differences. We found no significant effect of students' interest level, engagement with optional exercises, or use of large language models on their perception of feedback usefulness. We also explore student preferences for future feedback features, finding a significant preference for showing the differences between generated and expected test outputs.

## 1 Introduction

Giving timely and formative feedback to students is very important for the learning process [11, 28]. However, doing so in large courses is difficult [19] and essentially impossible in cases such as Massive Open Online Courses (MOOCs). Over the years, many automated assessment frameworks have been proposed to address this problem. Automated assessment tools such as MOOSHAK/ENKI [20] or GITSEED [18], among others [6, 10, 22, 26], allow faculty to automatically test students' submissions and provide them with varying levels of feedback.

Many newer automated assessment systems focus on (1) making it more usable for students by targeting well-known interfaces such as GIT, and (2) making it easier to maintain and modify by faculty. Such tools, like GITSEED, allow faculty to fully customize the feedback returned to users. This makes it easy to leverage state-of-the-art software analysis and automated diagnosis tools, which in turn allow students to find and repair their mistakes more quickly.

However, despite these advances, significant challenges remain in the context of logic programming. Novice students usually find it hard to debug logic programs. This is mainly due to the lack of traditional control and data flows they learn in imperative languages. In this work, we analyze these difficulties and students' behavior when learning Prolog. To accomplish this, we created PROHELP, an automated

To appear in EPTCS.

assessment platform for Prolog built on top of GITSEED. PROHELP includes feedback features ranging from automated testing to typo-correction suggestions and validation of implementation techniques.

We designed and deployed a mixed-methods study with the students who interacted with PROHELP. We used survey responses collected during a 9-week undergraduate course with 365 students. Overall, we analyzed 144 survey responses.

The main contributions of this work are:

- Understanding which types of feedback implemented in PROHELP students find the most helpful;
- Gauging students' interest in more complex types of automated feedback they would like to see in the future.

Results show that the feedback provided to students was helpful and that the PROHELP automated assessment system is user-friendly. Students also showed great interest in proposed improvements and suggested some themselves.

This paper is organized as follows: in Section 2 we present an overview of related work. Following that, in Section 3, we present the course in which this study was conducted, and in Section 4, we describe the automated assessment system and the types of feedback provided. Then, in Section 5, we present the methodology for our study, and in Section 6, we present the analysis and results. Finally, in Section 7, we present a discussion on the obtained results and conclude the paper.

## 2 Related Work

**Automated Assessment Tools (AATs).** Automated assessment of programming assignments has a long history of more than 60 years of research [19]. Most AATs assess programs by either comparing them with a known-correct solution or by executing a series of tests and checking their results [11]. A large number of current AATs are based around Integrated Development Environment-like web-interfaces with examples like CODEOCEAN [26], MOOSHAK/ENKI [14, 20] and WEB-CAT [6]. Other platforms, like PROGEDU [3], GitHub Classroom[1], and GITSEED [18] focus on Git, taking advantage of Continuous Integration (CI) actions to assess students' work as it is submitted to the repository. These tools can be easier for students to use because they are often already familiar with the Git workflow.

**Automated Feedback for Logic Programming.** Many systems have been proposed for automated feedback, debugging, and repair of logic programming languages [13, 16, 27, 1]. These tools face different challenges compared with other programming paradigms due to the declarative nature of the languages, lack of explicit control flow, and issues like non-determinism, non-monotonic reasoning, or non-termination [16]. INCOM [13] introduces a two-stage tutoring model in which students first engage in task analysis, then implement Prolog predicates. This structured approach assists learners in bridging the gap between problem specification and algorithm development. Meanwhile, PRAM [16] focuses on the automatic assessment of Prolog programs by incorporating both static and dynamic metrics to evaluate code style, complexity, and dynamic correctness, thereby reducing instructor workload and facilitating the shift from procedural to declarative thinking. Complementing these educational tools, PROFL [27] applies fault localization techniques, particularly coverage-based and mutation-based approaches, to identify errors in Prolog programs effectively. Moreover, the FORMHE [1] system extends automated feedback to Answer Set Programming by combining traditional fault localization methods

[1] `https://classroom.github.com/`

with modern strategies such as Large Language Model (LLM)-based fault localization and repair and mutation-based program repair techniques.

One common element of earlier works on automated logic programming feedback is the use of structured, rule-based feedback, which can force students to solve problems in a certain way. Meanwhile, more recent approaches follow a test-based fault localization method, which helps students identify problems in their programs rather than guiding them in a particular direction. However, these newer tools lack comprehensive user studies to evaluate their performance. Our work addresses this research gap by directly engaging with the target audience of the tool – the students – and prioritizing their needs.

## 3 Course Structure

We evaluated students' behavior during a 9-week-long bachelor's level logic programming course. During this course, students learned classic logic reasoning, as well as logic programming through Prolog. Overall, students faced three different types of Prolog exercises during this course: a 5-part role-playing exercise to familiarize students with the programming language, 22 optional recitation exercises, and one graded and mandatory final project. The mandatory project was worth 50% of the final grade, while the remaining 50% was from other theoretical assessments. Next, we describe the Prolog exercises in more detail.

- **Role-playing Exercise**
  - Optional, not graded, online only
  - 5 puzzles over 5 days
  - A series of simple logic puzzles where students help a detective catch a group of criminals using logic programming;
- **Recitation Exercises**
  - Optional, not graded, partially solved in-class using pen and paper
  - 22 exercises
  - 4 Prolog exercise sheets grouped into the topics of: lists, arithmetic, negation, and higher-order functions;
- **Project**
  - Mandatory, graded
  - 5 weeks duration
  - Worth 50% of the final course grade. The goal of this year's project was to create a solver for the logic puzzle *star battle*;

In Figure 1, we show the timeline of the different exercises solved by the students. During the first weeks, the students learned logic fundamentals. Then, during week 3, the students started learning Prolog, with the optional exercise sets being released during weeks 3, 4, and 5. Meanwhile, the project statement was released at the beginning of week 4 with the deadline at the end of week 9.

## 4 PROHELP: Prolog Automated Assessment Platform

The submission and evaluation platform used in this course was PROHELP, an extension built on top of the open-source automated assessment framework GITSEED [18]. GITSEED provides students with

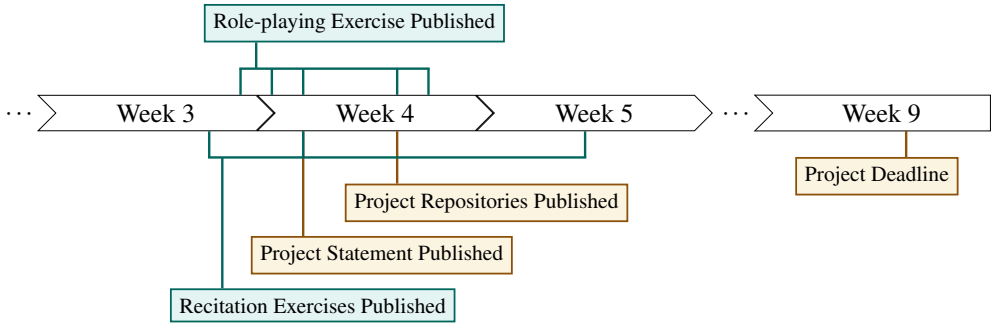



Figure 1: Course exercises timeline.

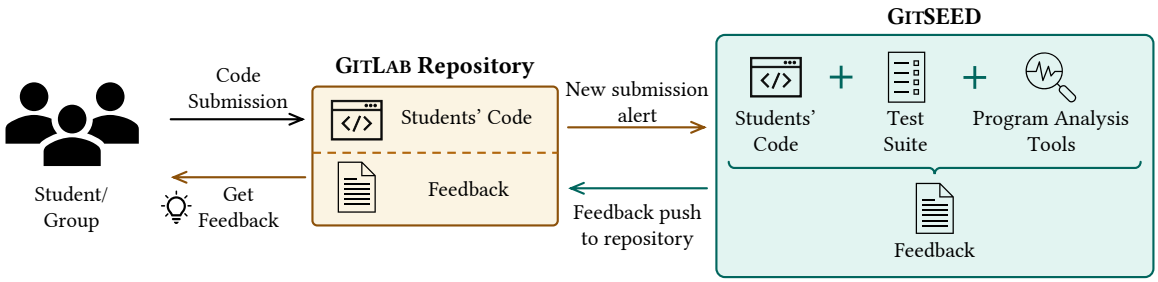



Figure 2: Simplified overview of GitSEED's operation scheme.

personalized feedback on programming assignments while they learn Git fundamentals. Unlike traditional assessment platforms, GitSEED operates within GitLab's CI workflow, eliminating the need for students to learn new interfaces. As Figure 2 shows, when students submit their work to GitLab, GitSEED automatically evaluates it using a predefined test suite and program analysis tools specified by the faculty. The evaluation report is then pushed directly to the students' git repositories, ensuring immediate access to personalized feedback.

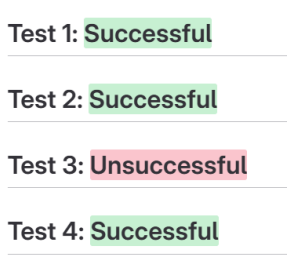
### 4.1 Automated Feedback

The minimum configuration of GitSEED simply runs a test suite on the student's submission and reports the results as feedback. However, as shown in Figure 2, GitSEED can be extended with custom program-analysis tools to provide students with more personalized feedback. Building on GitSEED's standard automatic testing (a) and score rankings (e), we created ProHelp to implement five additional types of feedback specific to the Prolog context. Below, we describe each of the 7 types of automated feedback, along with an example for each one, shown in Figure 3.

a) **Automatic testing:** This is the basic feature consisting of running the test suite and reporting the results. The faculty can decide to show the body of failing test cases or not.

b) **Predicate scoring:** For graded assignments, each predicate $p$ that the students need to implement gets assigned X points by the faculty. If a student passes the tests for $p$, then they get X points.

c) **Syntax error highlighting:** ProHelp highlights snippets of code with syntax errors, which makes them easier to locate compared with the line and column coordinates provided by the Prolog interpreter.

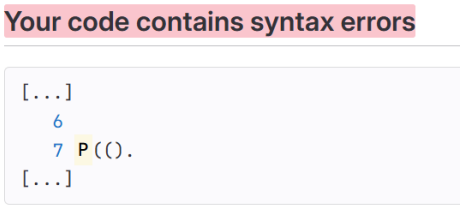



(a) Automatic testing

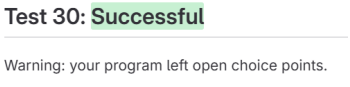
Scores

| Predicate | Current Score | Max Score |
|---|---|---|
| pred1/1 | 0.50 | 0.50 |
| pred2/1 | 0.50 | 0.50 |
| pred3/3 | 0.00 | 1.00 |
| Total | 1.00 | 2.00 |

(b) Predicate Scoring



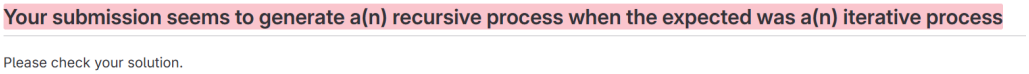

(c) Syntax error highlighting



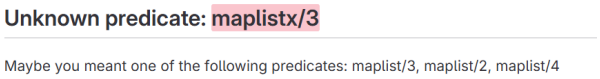

(d) Open choice points

| | Group ID | Total Score | Successful Tests | Unsuccessful Tests | Number of Submissions | Days Since Project Kick-off |
|---|---|---|---|---|---|---|
| 0 | studentId1 | 16 | 38 | 0 | 7 | 16 |
| 1 | studentId2 | 4.5 | 20 | 18 | 32 | 16 |

(e) Score rankings




(f) Solution type validation




(g) Unknown predicate name suggestions

Figure 3: Screenshots of the different types of student feedback available in PROHELP.

d) **Open choice points warning:** Prolog allows enumerating multiple answers for a given query. However, students often forget this fact. This can result in the first answer being correct with respect to the specification, but one of the following answers being incorrect. When a predicate terminates with an open choice point (indicating there are more possible answers), we warn students so they can verify whether this is intended behavior.

e) **Score rankings:** Each assignment has a global leaderboard that shows the scores and number of passed tests for each student. This incentivizes some students to compete and solve the most exercises as early as possible.

f) **Solution type validation:** Some of the assignments ask students to solve problems using specific techniques (for example, using recursion). In these assignments, we use a heuristic to determine if the student's program is using the correct method or not. The heuristic inspects the call dependencies of the main predicate: it is classified as *functional* if it relies on built-in higher-order predicates (such as `maplist` or `foldl`), *recursive* if it depends on itself, and *iterative* if it delegates the work to an auxiliary predicate of higher arity (typically carrying an accumulator).

g) **Unknown predicate name suggestions:** When a student uses a predicate that is undefined, PROHELP suggests other predicates with similar names if they exist.

### 4.2 Future Directions

One of the goals of this work is to gauge students' opinions on the feedback features already implemented and decide which ones to prioritize for future work. Planned features include several AI-based procedures using formal methods targeted to the specific context of logic programming.

- **Fault localization:** Identifying what section of the program contains bugs could help students narrow their focus. The granularity of the bug location could also be controlled by the faculty, such as identifying a specific predicate or a specific set of lines.
- **Bug fix suggestions:** The bug identification can be taken one step further and also include a description of the type of bug and/or a hint on how to fix it. One way to accomplish this is to use automated program repair to find a bug fix, then combine the buggy and corrected programs to generate a hint.
- **Infinite loop identification:** Prolog programs can suffer from infinite loops due to the depth-first search algorithm used by default in many Prolog interpreters. Helping students identify which sections of the program are responsible for the infinite loop behavior can be a helpful tool for students.
- **Test output differences:** The test suites in PROHELP are composed of unit tests implemented through `plunit`. Almost always, these tests consist of a series of predicate calls followed by an assertion that compares a value with the expected response. It could be useful for students to provide more information when these assertions fail, particularly showing the difference between generated and expected values.

## 5 Methodology

PROHELP was deployed in a BSc-level class on logic reasoning and programming with 365 students at Universidade de Lisboa, Portugal, during the 2024 academic year. Students were given automated feedback on 3 different types of Prolog assignments: a role-playing exercise, recitation exercises, and a final project.

To understand the strengths and shortcomings of PROHELP, we performed a post-experience survey. The survey, which is replicated in Appendix A, had an estimated duration of 15 minutes and was performed using EUSurvey[2] during the two weeks following the last programming assignment.

Of the 312 students who interacted with PROHELP during the course, we obtained 148 survey responses (47%), of which we dropped 4 due to failed attention checks [17]. Of the 144 valid responses, 28 respondents identified as female (19%), 111 as male (77%), 1 as other (<1%), and 4 preferred not to disclose (3%). While we have no information on the gender distribution of students who took this class, the official distribution of enrolled students in this BSc degree is 83% male and 17% female. This provides some confidence that our sample is representative of the overall course population.

In this work, we plan to answer the following three research questions:

**RQ1.** How valuable did students find the feedback, and which aspects of it were the most useful?

**RQ2.** Does student interest, engagement or LLM-usage affect how helpful they found the feedback?

**RQ3.** What features do students believe would be most beneficial in the future?

[2] `https://ec.europa.eu/eusurvey`

### 5.1 Data Analysis

Part of our study focuses on analyzing 3- and 5-point Likert scales regarding the usefulness of different aspects of the feedback received. For RQ1 and RQ3, we are interested in how a student's different answers compare to one another (the ratings for the different types of feedback). To accomplish this, we use the Friedman rank sum test [7] to check for any statistically significant differences between the feedback types. In cases where there are such differences, we employ the post-hoc pairwise Durbin-Conover test [4] with the Holm correction method [9], in order to identify which pairs of feedback types have statistically significant differences in usefulness.

In RQ2, we are interested in how different sub-groups of students rated the usefulness of PROHELP (on a 5-point Likert scale). To accomplish this, we use different tests depending on the nature of the independent variable. Students' interest was measured on a scale from 1 to 5 (an ordinal variable) and, as such, we use Spearman's rank correlation [25] to compare it with the usefulness. Meanwhile, students' engagement and usage of LLMs were both measured through categorical variables. We use the Kruskal-Wallis test [8] for engagement, since it has more than 2 categories, and the Mann-Whitney $U$ test [15] for LLM usage, since it has only 2 categories. The significance level for all tests was set at 0.05.

The survey also included two open-ended questions about future feedback suggestions and general opinions on the system. The answers were coded according to a unified code-book for Feedback and Improvements. The code-book was created by the first author. The answers were coded independently by the first and second authors, with any differences reconciled through a subsequent discussion.

### 5.2 Ethics and Privacy

All students had access to the same system and the same types of feedback. This ensured that no group of students had any advantage over the rest. Participation in the survey was voluntary, unremunerated, and all participants provided informed consent for data collection and processing before the survey. Participants were recruited through a course announcement after the last assignment. No identifiable personal data was collected. At Universidade de Lisboa, where the course was held and the survey was conducted, studies with these characteristics do not need to be submitted to the Ethics Committee (as confirmed by the Chair of the Ethics Committee). Co-authors from institutions other than Universidade de Lisboa had access only to aggregate statistics, not to individual responses. We have completed the European Commission's ethics self-assessment checklists and comply with all the guidelines.

## 6 Results

This work aims to understand the advantages and limitations of the PROHELP system and explore what types of feedback students would like to have in the future. The statistical analysis and plots presented in this section were created using R version 4.4.2 [23], statsExpressions [21], ggstats [12] and ggsignif [5].

### 6.1 Usefulness and usability of the feedback (RQ1)

To gauge if PROHELP was useful to students, we asked them if they agreed with the statement "I think the automated feedback helped increase my grade in this course" using a 5-point Likert scale. Figure 4 shows that 74% of student responses agreed that the provided feedback was useful, while 11% disagreed and 15% neither agreed nor disagreed. This indicates that PROHELP was helpful for students. We also performed a usability analysis through the System Usability Scale (SUS) questionnaire [2]. The average

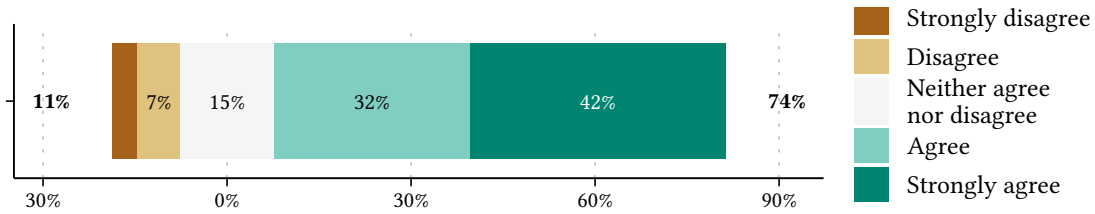



Figure 4: Agreement distribution for the statement "I think the automated feedback helped increase my grade in this course.".

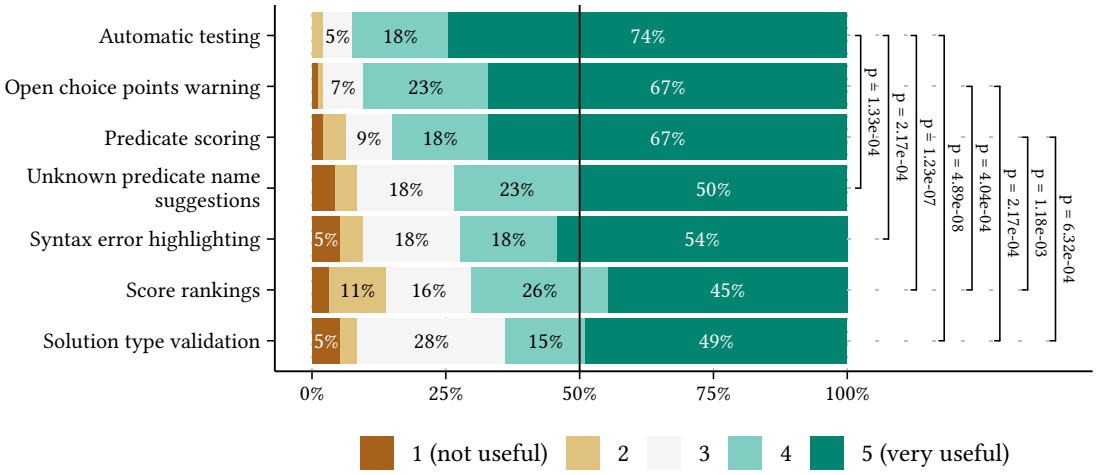



Figure 5: Usefulness rating of the different types of feedback. The median response for each type is shown by the vertical line. The right side of the figure shows pairs of feedback types for which the Durbin-Conover post-hoc test indicates statistically significant differences in ratings. For example, there is a statistically significant difference between the responses for "Automatic testing" and "Score rankings" with $p = 1.23\text{e-}07$.

SUS score among our respondents was 78.5 which corresponds to a B+ grade according to the curved grading interpretation by Sauro and Lewis [24]. While there is room for improvement, the results show that our system is fairly usable in its current state.

This study also aims to understand if any type of feedback stood out as particularly more useful than the others. Hence, we asked the students to rank each of the 7 types of feedback in Section 4 on a scale from 1 (not useful) to 5 (very useful). Figure 5 presents the answer distribution to these questions. Most students ranked all 7 types of feedback as useful, with all of the medians being $\geq 4$.

Since not all students interacted with all forms of feedback and the Friedman test requires a full block design, we excluded students who did not answer all 7 questions, resulting in 94 samples. The Friedman test allows us to reject the null hypothesis of there being no significant difference between the different types of feedback ($\chi^2(6) = 58.66,\ p = 8.436 \times 10^{-11},\ \hat{W}_{Kendall} = 0.10$). As such, to discover which types of feedback were significantly more useful, we used the post-hoc pairwise Durbin-Conover test. The right side of Figure 5 shows the pairs of feedback types for which there is a significant difference in median ranks, along with the respective p-values. These statistically significant tests show that "Automatic testing" was the most helpful type of feedback, being more useful than 4 other types, and that "Open choice points warning" and "Predicate scoring" were more useful than 2 other types each.

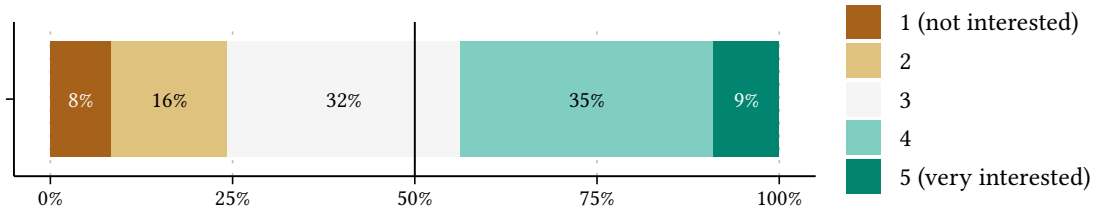



Figure 6: Answer distribution for the question "How interested are you in the topics of logic and logic programming?". The median response is shown by the black vertical line.

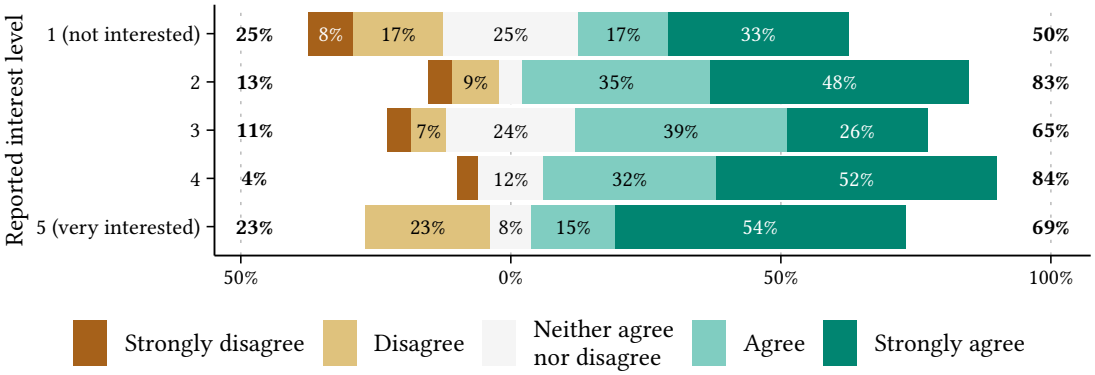



Figure 7: Effect of interest in the agreement distribution for the statement "I think the automated feedback helped increase my grade in this course.".

Meanwhile, the least helpful types of feedback were "Score rankings" and "Solution type validation", with each being less useful than 3 other types.

**A1.** Most students consider that the provided feedback was helpful, and the system has a usability score of B+. The most useful feedback were "Automatic testing", "Open choice points warning" and "Predicate scoring", while the least useful were "Score ranking" and "Solution type validation". These conclusions are supported by statistically significant differences.

## 6.2 Impact of interest, engagement, and LLM use (RQ2)

Another goal of this study is to determine if students' interest, engagement with optional exercises, and having used LLMs (or not) had any effect on their perception of the usefulness of the feedback.

### 6.2.1 Interest

We asked students to rate their interest in logic and logic programming on a scale from 1 to 5. The answers, summarized in Figure 6, show a fairly normal distribution, with most students having some interest in the topics and a small number showing either no interest or a high level of interest.

To test our hypothesis that more interested students take better advantage of automated feedback, we compared students' interest with the level of agreement with the statement "I think the automated feedback helped increase my grade in this course". Figure 7 shows the agreement responses grouped by interest level. A Spearman's rank correlation analysis does not allow us to reject the null hypothesis,

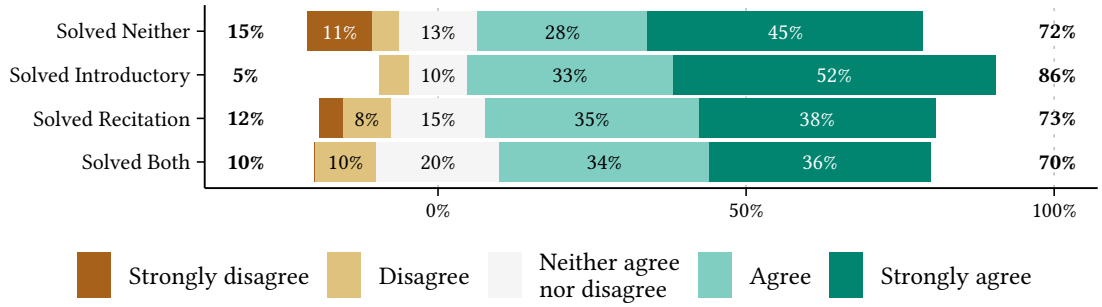



Figure 8: Effect of engagement in the agreement distribution for the statement "I think the automated feedback helped increase my grade in this course.".

| **LLM Tool** | ChatGPT | Copilot | Claude | Gemini | Blackbox | Deepseek | Perplexity | ZZZ Code |
|---|---|---|---|---|---|---|---|---|
| **Count** | 82 | 16 | 5 | 2 | 1 | 1 | 1 | 1 |

Table 1: Number of students that reported using each LLM tool.

which means there is no statistically significant relationship between students' interest level and how beneficial they found the feedback received ($S = 4.26 \times 10^5$, $p = 0.08$, $\hat{\rho} = 0.14$).

### 6.2.2 Engagement

Students had the opportunity to use ProHelp with two types of optional exercises: a set of 5 introductory exercises and a set of 22 recitation exercises. Considering the 144 respondents to our survey, 47 did not solve any optional exercises, 50 solved at least one exercise from both optional sets, 21 solved only exercises from the introductory set, and 26 solved only exercises from the recitation set. Our hypothesis is that more engaged students (ones who solved optional exercises) would have taken better advantage of ProHelp and thus found it more useful. Figure 8 shows the relationship between the 4 engagement groups and the agreement levels with the statement "I think the automated feedback helped increase my grade in this course". To test our hypothesis, we used a Kruskal-Wallis test, which shows no evidence of there being a statistical difference between the four groups ($\chi^2(3) = 2.19$, $p = 0.53$, $\hat{\varepsilon}^2 = 0.02$).

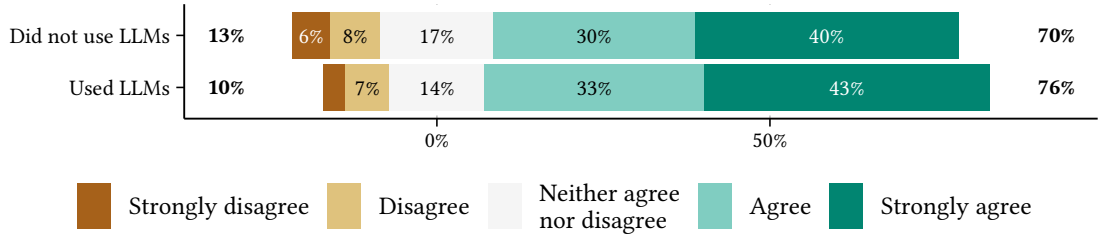



Figure 9: Effect of LLMs usage in the agreement distribution for the statement "I think the automated feedback helped increase my grade in this course.".

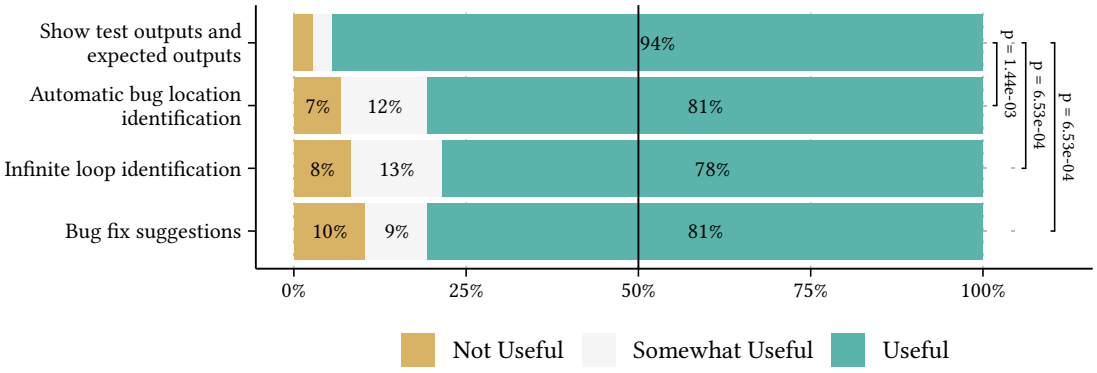



Figure 10: Usefulness ratings for the different types of feedback not yet implemented in PROHELP. The median response for each type of feedback is shown by the black vertical line. The right side of the figure shows pairs of feedback types for which the Durbin-Conover post-hoc test indicates statistically significant differences in ratings.

### 6.2.3 Usage of LLMs

Students were allowed to use Large Language Models (LLMs) during this course, as long as they disclosed that usage. Considering the 144 responses to our survey, 91 reported using LLMs when solving assignments, while 53 reported that they did not use LLMs. Table 1 shows the number of students who reported using each LLM tool/platform, with the most popular ones being ChatGPT and Copilot. Furthermore, 21 students reported using more than one LLM. We were interested in whether using LLMs affects how students perceive the usefulness of PROHELP (for example, students who used LLMs might have found PROHELP less useful because they already had personalized feedback from the LLM). In Figure 9, we show the usefulness ratings for students who used and did not use LLMs. According to the Mann-Whitney $U$ test, there is no statistical difference between the two groups ($W = 2261.50,\ p = 0.51,\ r = -0.06$).

**A2.** There is no evidence that interest level, having used the platform for optional exercises, or having used LLMs, affects how helpful students found PROHELP overall.

## 6.3 Future directions (RQ3)

Our last goal is to better understand which features and improvements to the feedback students would most like to see in the future, while also analyzing which aspects of the course students did not like. In the survey, we asked students to rate the 4 types of (not yet implemented) feedback described in Section 4.2 between Not Useful, Somewhat Useful and Useful. Figure 10 shows the students' answers, where at least 75% of students rated each of the proposed feedback types as Useful. These responses hint that these are good goals to work towards.

To check if any type of feedback had a significant preference over another, we used a Friedman test, which rejects the null hypothesis ($\chi^2(3) = 20.74,\ p = 1.19 \times 10^{-4},\ \hat{W}_{Kendall} = 0.05$). Next, to check which types of feedback students think will be more useful, we used the post-hoc pairwise Durbin-Conover test. The right side of Figure 10 shows the pairs of feedback types that showed statistical significance. Students rated "Showing the differences between generated and expected output" as significantly

more useful than the other 3 types of feedback.

At the end of the survey, we asked students to suggest other types of feedback they would like to see in the future and to provide general comments/criticisms. One common criticism among students was that the unit tests were not specific enough, i.e., each test covered too much functionality. For example, P-31 said: "The project tests were not exhaustive enough.". This left students with little information when tests failed and highlights the importance of faculty creating good unit tests that assess each functionality in isolation. There were also other comments regarding the course organization and grading method, which we have relayed to the faculty.

One interesting suggestion from some students concerns the qualitative component of the project grade, which is manually assigned and encompasses characteristics such as code readability, organization, and documentation. Students expressed a need for an estimate of this qualitative component in the automated feedback, even if the final score remained hand-assigned by faculty.

**A3.** Although students found all proposed types of feedback useful, there is a significant preference for the "Show the outputs and expected outputs" compared with the remaining feedback types. Students would like an automated estimate for the qualitative component of their projects.

## 7 Discussion and Conclusion

This work analyzes students' behaviors during a 9-week-long undergraduate class on logic programming. Specifically, we study how they interacted with an automated assessment platform for Prolog code. This platform, named PROHELP, included several types of automated feedback to help students find and fix their bugs more easily. Our main contribution is the analysis of a student survey with 144 valid responses, which gathers insights into students' preferences and complaints regarding PROHELP.

Survey analysis indicates that users found the feedback received useful and the PROHELP interface was user-friendly. For instance, P-101 said *"I loved the assessment system. It was very efficient and helpful. From what I have heard from previous years, it was a huge improvement"*. The components of PROHELP that students most appreciated include automatic testing, warnings about open choice points, and scoring for each required predicate. Students also displayed great interest in the more advanced types of feedback we proposed and suggested some themselves. These findings will be valuable for future research, enabling educators and developers to focus on the most impactful types of feedback.

Results show no statistically significant relationship between students' perception of feedback usefulness and their interest, engagement, or usage of LLMs. Even so, there is a small but not statistically significant correlation ($p = 0.08$) between students' interest and their usefulness rating, suggesting that further research – with a larger sample or a different methodology – is warranted. The findings on LLM usage are particularly intriguing, and we propose several possible explanations for it: (1) students underreported LLM-usage due to its negative connotation, (2) the feedback received in PROHELP is orthogonal to the questions students usually ask LLMs, or (3) LLMs are not (yet) very useful for Prolog. Further investigation is needed to determine which, if any, of these factors apply.

Overall, our findings demonstrate that automated feedback is a valuable resource for students learning Prolog, and that even simple feedback mechanisms, when well integrated into familiar workflows, can have a meaningful impact on the learning experience. These insights can guide educators and tool developers in prioritizing the types of feedback that students find most impactful, ultimately contributing to more effective logic programming education.

## Acknowledgments

This work supported by Portuguese national funds through FCT, under project 2023.14280.PEX (DOI: 10.54499/2023.14280.PEX). This work was also supported by grant PID2022-139835NB-C21 funded by MCIN/AEI/10.13039/501100011033 and by ERDF, EU; and by HORIZON-MSCA-2025-PF, project 101269051 — Sherlock4Py funded by REA, EU. This work was additionally supported by the Carnegie Mellon University Portugal Program and FCT under grants PRT/BD/152086/2021 (DOI: 10.54499/-PRT/BD/152086/2021) and PRT/BD/153739/2021 (DOI: 10.54499/PRT/BD/153739/2021). This work was also partially supported by the National Science Foundation (NSF) under Award CCF2427581 and DARPA under Agreement FA8750-24-9-1000.

## References

[1] Ricardo Brancas, Vasco Manquinho & Ruben Martins (2025): *Combining Logic and Large Language Models for Assisted Debugging and Repair of ASP Programs*. In: *IEEE Conference on Software Testing, Verification and Validation, ICST 2025, Napoli, Italy, March 31 - April 4, 2025*, IEEE, pp. 646–657, doi:10.1109/ICST62969.2025.10988950.

[2] J. B. Brooke (1996): *SUS: A 'Quick and Dirty' Usability Scale*. In: *Usability Evaluation In Industry*, CRC Press, pp. 207–212.

[3] Hsi-Min Chen, Wei-Han Chen, Nien-Lin Hsueh, Chi-Chen Lee & Chia-Hsiu Li (2017): *ProgEdu - an automatic assessment platform for programming courses*. In: *2017 International Conference on Applied System Innovation (ICASI)*, pp. 173–176, doi:10.1109/ICASI.2017.7988376.

[4] W.J. Conover (1999): *Practical Nonparametric Statistics*. Wiley Series in Probability and Statistics, Wiley.

[5] Ahlmann-Eltze Constantin & Indrajeet Patil (2021): *ggsignif: R Package for Displaying Significance Brackets for 'ggplot2'*. *PsyArxiv*, doi:10.31234/osf.io/7awm6. Available at `https://psyarxiv.com/7awm6`.

[6] Stephen H. Edwards & Manuel A. Pérez-Quiñones (2008): *Web-CAT: automatically grading programming assignments*. In June Amillo, Cary Laxer, Ernestina Menasalvas Ruiz & Alison Young, editors: *Proceedings of the 13th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education, ITiCSE 2008, Madrid, Spain, June 30 - July 2, 2008*, ACM, "", p. 328, doi:10.1145/1384271.1384371.

[7] Milton Friedman (1937): *The Use of Ranks to Avoid the Assumption of Normality Implicit in the Analysis of Variance*. *Journal of the American Statistical Association* 32, pp. 675–701. Available at `https://doi.org/10.2307/2279372`.

[8] M. Hollander, D.A. Wolfe & E. Chicken (2013): *Nonparametric Statistical Methods*. Wiley Series in Probability and Statistics, Wiley. Available at `https://doi.org/10.1002/9781119196037`.

[9] Sture Holm (1979): *A Simple Sequentially Rejective Multiple Test Procedure*. *Scandinavian Journal of Statistics* 6, pp. 65–70. Available at `https://www.jstor.org/stable/4615733`.

[10] Callum Iddon, Nasser Giacaman & Valerio Terragni (2023): *GRADESTYLE: GitHub-Integrated and Automated Assessment of Java Code Style*. In: *45th IEEE/ACM International Conference on Software Engineering: Software Engineering Education and Training, SEET@ICSE 2023, Melbourne, Australia, May 14-20, 2023*, IEEE, "", pp. 192–197, doi:10.1109/ICSE-SEET58685.2023.00024.

[11] Hieke Keuning, Johan Jeuring & Bastiaan Heeren (2019): *A Systematic Literature Review of Automated Feedback Generation for Programming Exercises*. *ACM Trans. Comput. Educ.* 19(1), pp. 3:1–3:43. Available at `https://doi.org/10.1145/3231711`.

[12] Joseph Larmarange (2025): *ggstats: Extension to 'ggplot2' for Plotting Stats*. Available at `https://larmarange.github.io/ggstats/`. R package version 0.8.0, https://github.com/larmarange/ggstats.

[13] Nguyen-Thinh Le & Niels Pinkwart (2011): *INCOM: A web-based homework coaching system for logic programming*. In: *Conference on Cognition and Exploratory Learning in Digital Age*, Citeseer, pp. 43–50.

[14] José Paulo Leal & Fernando M. A. Silva (2003): *Mooshak: a Web-based multi-site programming contest system*. *Softw. Pract. Exp.* 33(6), pp. 567–581. Available at https://doi.org/10.1002/spe.522.

[15] H. B. Mann & D. R. Whitney (1947): *On a Test of Whether one of Two Random Variables is Stochastically Larger than the Other*. *The Annals of Mathematical Statistics* 18(1), pp. 50 – 60. Available at https://doi.org/10.1214/aoms/1177730491.

[16] Fatima Z. Mansouri, Cleveland Augustine Gibbon & Colin A. Higgins (1998): *PRAM: prolog automatic marker*. In Gordon Davies & Mícheál Ó'héigeartaigh, editors: *Proceedings of the 6th Annual Conference on the Teaching of Computing and the 3rd Annual SIGCSE Conference on Innovation and Technology in Computer Science Education, ITiCSE 1998, Dublin City University, Ireland, 18-21 August 1998*, ACM, pp. 166–170, doi:10.1145/282991.283108.

[17] Daniel M. Oppenheimer, Tom Meyvis & Nicolas Davidenko (2009): *Instructional manipulation checks: Detecting satisficing to increase statistical power*. *Journal of Experimental Social Psychology* 45(4), pp. 867–872. Available at https://doi.org/10.1016/j.jesp.2009.03.009.

[18] Pedro Orvalho, Mikolás Janota & Vasco Manquinho (2024): *GitSEED: A Git-backed Automated Assessment Tool for Software Engineering and Programming Education*. In Mohsen Dorodchi, Ming Zhang & Stephen Cooper, editors: *Proceedings of the 2024 ACM Virtual Global Computing Education Conference V. 1, SIGCSE Virtual 2024, Virtual Event, NC, USA, December 5-8, 2024*, ACM. Available at https://doi.org/10.1145/3649165.3690106.

[19] José Carlos Paiva, José Paulo Leal & Álvaro Figueira (2022): *Automated Assessment in Computer Science Education: A State-of-the-Art Review*. *ACM Trans. Comput. Educ.* 22(3), pp. 34:1–34:40. Available at https://doi.org/10.1145/3513140.

[20] José Carlos Paiva, José Paulo Leal & Ricardo Alexandre Peixoto de Queirós (2016): *Enki: A Pedagogical Services Aggregator for Learning Programming Languages*. In Alison Clear, Ernesto Cuadros-Vargas, Janet Carter & Yván Túpac, editors: *Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education, ITiCSE 2016, Arequipa, Peru, July 9-13, 2016*, ACM, pp. 332–337. Available at https://doi.org/10.1145/2899415.2899441.

[21] Indrajeet Patil (2021): *statsExpressions: R Package for Tidy Dataframes and Expressions with Statistical Details*. *J. Open Source Softw.* 6(61), p. 3236. Available at https://doi.org/10.21105/joss.03236.

[22] Matthew Peveler, Jeramey Tyler, Samuel Breese, Barbara Cutler & Ana L. Milanova (2017): *Submitty: An Open Source, Highly-Configurable Platform for Grading of Programming Assignments (Abstract Only)*. In Michael E. Caspersen, Stephen H. Edwards, Tiffany Barnes & Daniel D. Garcia, editors: *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education, SIGCSE 2017, Seattle, WA, USA, March 8-11, 2017*, ACM, "", p. 641, doi:10.1145/3017680.3022384.

[23] R Core Team (2024): *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. Available at https://www.R-project.org/.

[24] Jeff Sauro & James R Lewis (2016): *Quantifying the user experience: Practical statistics for user research*. Morgan Kaufmann. Available at https://doi.org/10.1016/C2010-0-65192-3.

[25] Charles Spearman (1904): *The proof and measurement of association between two things*. *The American Journal of Psychology* 15(1), pp. 72–101. Available at https://doi.org/10.2307/1412159.

[26] Thomas Staubitz, Hauke Klement, Ralf Teusner, Jan Renz & Christoph Meinel (2016): *CodeOcean - A versatile platform for practical programming excercises in online environments*. In: *2016 IEEE Global Engineering Education Conference, EDUCON 2016, Abu Dhabi, United Arab Emirates, April 10-13, 2016*, IEEE, "", pp. 314–323, doi:10.1109/EDUCON.2016.7474573.

[27] George Thompson & Allison K. Sullivan (2020): *ProFL: a fault localization framework for Prolog*. In Sarfraz Khurshid & Corina S. Pasareanu, editors: *ISSTA '20: 29th ACM SIGSOFT International Symposium on Software Testing and Analysis, Virtual Event, USA, July 18-22, 2020*, ACM, pp. 561–564. Available at https://doi.org/10.1145/3395363.3404367.

[28] Benedikt Wisniewski, Klaus Zierer & John Hattie (2020): *The power of feedback revisited: A meta-analysis of educational feedback research*. Frontiers in psychology 10, p. 487662.

# A Survey Protocol

The following survey was designed to collect anonymous feedback from undegraduate students enrolled in the *Logic Programming* course regarding their experience with the automated feedback system provided via GitLab. The survey was estimated to take approximately 10 to 15 minutes. Several questions were included to assess usability, usefulness, and pedagogical effectiveness of the feedback tools. Some questions functioned as attention checks. The questions and answers presented here were translated from Portuguese.

### Section 1: Demographics and Background

- **What gender do you identify with?**
  - Female
  - Male
  - Prefer not to say
  - Prefer to self-describe
- **What is your level of interest in the topics covered in the Programming Languages course? (Logic, Prolog)**

  Rating scale from 1 (not interested) to 5 (very interested)
- **During this course, did you experience difficulties using Git/GitLab? Select all that apply:**
  - Merge conflicts
  - SSH key problems
  - Problems interacting with VS Code
  - I had no problems
  - Other

### Section 2: Automated Feedback in Prolog

Students were reminded of the types of automated feedback they encountered on GitLab:

- Automatic test verification
- Predicate scoring
- Validation of solution types
- Dashboards with scores
- Syntax error highlighting
- Suggestions for incorrect predicate names
- Warnings about open choice points

Students were asked to rate their agreement with the following System Usability Survey Items using a 5-point Likert scale (1 = Strongly Disagree, 5 = Strongly Agree):

- I think that I would like to use this system frequently.
- I found the system unnecessarily complex.
- I thought the system was easy to use.
- I think that I would need the support of a technical person to be able to use this system.
- I found the various functions in this system were well integrated.
- I thought there was too much inconsistency in this system.
- I would imagine that most people would learn to use this system very quickly.
- I found the system very cumbersome to use.
- I felt very confident using the system.
- I needed to learn a lot of things before I could get going with this system.

Additionally, students rated the usefulness of each feedback component on a scale from 1 (Not useful at all) to 5 (Very useful):

- Suggestions for incorrect predicate names
- Syntax error highlighting
- Dashboards with scores
- Select "Not useful at all" in this line **[attention check]**
- Automatic test verification
- Predicate scoring
- Warning for open choice points
- Solution type validation (iterative, recursive, functional)

### Section 4: Role-playing Exercise

- **Did you solve the role-playing exercise?**
  - No
  - I started but did not finish
  - Yes
- **Why did you not solve/abandon/finish the exercises?** (Open text response)
- **Did you use LLMs (e.g., ChatGPT, Copilot) to help solve the role-playing exercise?**
  - No
  - Yes
- **Which LLM(s) did you use?** (Open text response)
- **What did you think of the sequence of puzzles from the role-playing exercise on a 5-point scale (1 = Not entertaining at all, 5 = Very entertaining)?**

### Section 5: Recitation Exercises

- **Did you solve the recitation exercises using Git/GitLab?**
  - No
  - I started but did not finish
  - Yes
- **Why did you not solve/abandon/finish the exercises?** (Open text response)
- **Did you use LLMs (e.g., ChatGPT, Copilot) to help with the recitation exercises?**
  - No
  - Yes
- **Which LLM(s) did you use?** (Open text response)

### Section 6: Project

- **Did you use LLMs (e.g., ChatGPT, Copilot) to help with the project?**
  - No
  - Yes
- **Which LLM(s) did you use?** (Open text response)

### Section 7: Final Reflections

- **Please rate the following statements on a 5-point Likert scale (1 = Strongly Disagree, 5 = Strongly Agree)**
  - I believe the automated feedback I received on GitLab improved my grade in this course.
  - I believe the role-playing exercise improved my grade in this course.
  - Select "Strongly Agree" to demonstrate you are paying attention. **[Attention check]**
  - I believe the recitation exercises on GitLab improved my grade in this course.
- **Please rate the potential utility of proposed features on a 3-point scale (Indifferent, Somewhat Useful, Useful):**
  - Suggestions for corrections on incorrect submissions
  - Showing program output for each test compared to expected output
  - Automatic identification of bug locations
  - Detection of infinite loops
- **Please suggest any other features you would like to see.** (Open text response)
- **Please leave any other feedback you may have (positive or negative).** (Open text response)