A Classroom Study of LLM-generated Feedback Intervention in Introductory Programming

Hasnain Heickal HHEICKAL@CS.UMASS.EDU
Andrew Lan ANDREWLAN@CS.UMASS.EDU
University of Massachusetts Amherst, MA, USA

Abstract

Large language models (LLMs) are increasingly used to provide automated feedback in introductory programming courses, yet empirical evidence from authentic classroom deployments comparing different feedback modalities remains limited. In this work, we present a large-scale classroom study in which AI-generated feedback was deployed through a randomized protocol in an introductory Python programming course. Students received one of three feedback conditions on incorrect submissions: natural language hints, AI-generated failing test cases, or no AI feedback. We release the resulting dataset, ProgFeed,[1] which captures 6,693 submissions from 215 consenting students across 17 labs, including feedback conditions, execution-based performance measures, and fine-grained temporal information. Using this data, we analyze learning trajectories, feedback quality, and submission behavior over repeated attempts. We find that natural language feedback is significantly associated with higher completion rates and faster convergence to correct solutions. Test case feedback, by contrast, exhibits heterogeneous effects that depend critically on feedback validity. Our results suggest that the form of AI-generated feedback matters, and that evaluating feedback quality—not just its presence—is essential for understanding its pedagogical impact.

Keywords: Automated Feedback, Programming Education, Large Language Models, Learning Analytics

1. Introduction

Feedback plays a central role in human learning, enabling learners to identify mistakes, refine strategies, and make progress toward mastery. In classroom settings, however, providing timely and individualized feedback is challenging: instructors must balance limited time and attention across large and heterogeneous student populations. Prior work has shown that immediate, just-in-time feedback Johnston (2015) and personalized feedback Kochmar et al. (2020) can significantly improve learning outcomes, yet such feedback is difficult to deliver consistently at scale. This challenge is particularly salient in introductory programming courses, where students require repeated guidance while debugging syntax and logic errors.

Automated feedback systems offer a promising avenue for scaling instructional support. Due to the structured nature of programming languages, a substantial body of work has explored automated feedback for programming tasks, including syntax error explanations, bug localization, and hint generation Keuning et al. (2018). More recently, advances in large language models (LLMs) have expanded the range of feedback that can be generated automatically, from fluent natural language explanations to dynamically constructed test

1. Publicly available at https://github.com/umass-ml4ed/progFeed-dataset-public.

cases Kazemitabaar et al. (2023); Kumar and Lan (2024b). As LLM-based feedback becomes increasingly integrated into real classrooms, a key open question is how different forms of AI-generated feedback affect student performance and learning over time.

Addressing this question requires data that captures not only student code but also the feedback received and how students respond across multiple attempts. Large-scale programming datasets such as CodeNet Puri et al. (2021) lack controlled feedback variation, making it difficult to disentangle feedback effects from confounding factors such as student ability or problem difficulty. Longitudinal datasets like Blackbox Brown et al. (2014) capture rich interaction traces, but without experimental manipulation. In this work, we address this gap directly.

We report on a classroom study in an introductory Python programming course in which AI-generated feedback was deployed through a randomized protocol and delivered *in situ* conditioned on each submission. The resulting dataset, ProgFeed, links complete submission histories to feedback conditions, execution outcomes, and temporal ordering, supporting analyses of short-term performance, time-to-success, and iteration behavior. Our contributions are:

- A classroom study enabling controlled comparison of natural language, failing test case, and no-AI feedback conditions in an authentic introductory programming course.
- ProgFeed, a longitudinal dataset capturing fine-grained submission trajectories, feedback conditions, and execution outcomes from this study.
- Empirical analyses demonstrating how the dataset supports rigorous study of learning dynamics and feedback effectiveness.

2. Related Work

2.1. Automated Feedback in CS Education

Automated feedback systems for introductory programming have traditionally focused on syntax errors and structural issues, leveraging compiler messages, static analysis, or unit tests Keuning et al. (2018). Early approaches provided next-step hints in the edit-distance space Paaßen et al. (2017) or learned mappings between program representations to generate feedback at scale Piech et al. (2015). With the rise of transformer-based models, research has increasingly focused on program repair and error explanations Yasunaga and Liang (2020, 2021); Leinonen et al. (2023). A key distinction is between *syntax errors*—studied extensively, with systems such as PyFiXV Phung et al. (2023a) demonstrating strong performance—and *logical errors*, which require understanding program semantics and are far less addressed by automated systems.

2.2. Logical Errors and LLM-Based Feedback

Recent work has begun exploring automated feedback for logical errors, particularly using LLMs. One common approach is failing test case feedback, which provides students with counterexamples to guide debugging Kumar and Lan (2024b). Other approaches generate Socratic questions Kumar and Lan (2024a) or concise natural language hints Phung

et al. (2023b,c). Several studies have evaluated LLM-generated feedback through expert annotation or small-scale user studies Phung et al. (2023b); Pankiewicz and Baker (2023), providing quality insights but typically examining a single modality, few problems, or limited participants—and rarely studying how feedback affects behavior across multiple attempts in authentic settings. Most closely related, Gabbay and Cohen (2024) combine LLM-generated explanations with test-based feedback in a programming MOOC and report improved correction rates and persistence; in contrast, we randomize *distinct* feedback modalities—natural language hints, failing test cases, or no AI feedback—within an authentic for-credit course and release the resulting dataset.

2.3. Evaluating Feedback Effectiveness

Evaluating the pedagogical impact of feedback remains challenging. Many studies rely on binary success metrics, though the *process* of learning—iteration patterns, time to resolution, error recurrence—is equally important Ahadi et al. (2018). Datasets such as CodeNet Puri et al. (2021) provide scale but lack instructional context, while longitudinal datasets like Blackbox Brown et al. (2014) capture traces without controlled interventions. ProgFeed bridges this gap by embedding randomized feedback interventions directly into data collection, linking complete submission trajectories to specific feedback conditions.

3. Dataset and Study Design

3.1. Course Context and Participants

We conducted an IRB-approved classroom study in an introductory Python programming course at a U.S. R1 institution during the Fall 2025 semester. Participation was limited to students aged 18 or older. Of the 365 students enrolled in the course, 220 consented to participate; the 215 consenters who submitted at least one solution constitute the released ProgFeed dataset. Students who did not consent received identical instructional experiences, including access to AI-generated feedback, but their data were excluded from all analyses and from the released dataset. The course primarily served first-year students, with additional enrollment from second- and third-year students. Most participants were majors in computer science, with representation from related disciplines such as mathematics, physics, and linguistics. The course met twice weekly for 75-minute lectures and included weekly laboratory assignments consisting of auto-graded programming problems.

AI-generated feedback was provided *only* for laboratory assignments. Quizzes, exams, pre-labs, and larger projects did not include AI feedback and are not part of the dataset. In addition to labs, students completed optional midweek pre-lab assignments that were structurally identical to labs and offered as extra-credit practice. Each pre-lab served as both a post-test for the previous lab and a pre-test for the upcoming lab, enabling within-course evaluation of learning across instructional units.

3.2. Data Collection and Experimental Design

ProgFeed contains all student submissions to lab and pre-lab problems. Students were allowed unlimited submissions per problem; each was evaluated against fixed instructor-authored test cases, always returning pass/fail verdicts. Feedback assignment was ran-

domized at the submission level within each lab; students were reassigned between labs to prevent systematic advantage. Feedback was *conditionally triggered* only on failing submissions, so exposure is not independent of correctness—all analyses are associative, not causal.

3.3. Collected Signals

ProgFeed is organized hierarchically across students, labs, problems, and submissions. For each submission, the dataset includes submitted source code, submission order within a student–problem sequence, fixed test case outcomes, execution-based performance metrics, and the assigned feedback condition and content when applicable. The dataset also includes auxiliary signals: a pre-course demographic survey, weekly post-lab surveys on perceived feedback usefulness, an end-of-course exit survey, and aggregate course performance measures such as lab scores, quiz scores, and final grades.

3.4. Privacy and De-identification

All personally identifiable information was removed prior to release. Student identifiers were replaced with deterministic pseudonymous identifiers generated using salted cryptographic hashes, preserving longitudinal structure while preventing re-identification. Automated scrubbing was applied to all code and text fields to remove residual identifying information such as names, email addresses, and file paths, while preserving program semantics.

3.5. Feedback Conditions and Generation

Each eligible (failing) submission was assigned to one of three conditions:

Group 0: No AI Feedback. Fixed test case pass/fail verdicts only.

Group 1: Failing Test Case Feedback. An AI-generated input–output pair designed to fail the student's submission.

Group 2: Natural Language Feedback. A concise natural language hint identifying a likely issue, without code suggestions or test cases.

All feedback was generated by `gpt-4o-mini` via the OpenAI API, given the problem description and student code. A smaller model was chosen for cost reasons given class size. Prompts were fixed across all labs; variation arose solely from student code.

Prompt for Failing Test Case Feedback (Group 1)

```
The student is given the problem and asked to implement the specified function(s).
You can see their code below.  Can you give a test case that would cause the
student's code to fail?  Only give the test case with the expected output.
Do not explain why it fails.  Do not include any markdown formatting.  If
the test case involves a list, use explicit values rather than variable names.
Problem:  ...  Code:  ...
```

Prompt for Natural Language Feedback (Group 2)

```
The student is given the problem and asked to implement the specified function(s).
You can see their code below.  Can you give a brief natural language hint
explaining why the code is failing?  Refer to relevant parts of the problem
description if appropriate.  Do not suggest code changes.  Do not give test
cases.  Focus on one important issue.
Problem:  ...  Code:  ...
```

3.6. Dataset Statistics

ProgFeed contains 6,693 submissions from 215 consenting students across 17 labs and 43 problems. Group 0 (no feedback): 4,067 submissions; Group 1 (test-case): 1,605; Group 2 (natural-language): 1,021. Students submitted multiple attempts per problem, supporting hierarchical analysis at student, problem, and lab levels.

4. Analysis

4.1. Post-Feedback Learning Dynamics

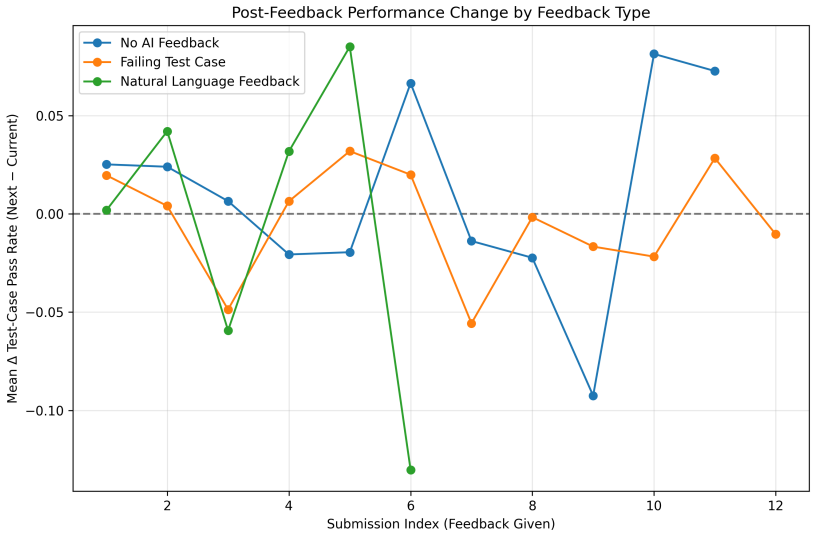

Figure 1: Mean change in test case pass rate by submission index, stratified by feedback condition.

We examine short-term learning dynamics by analyzing Δ pass rate—the change in test case pass rate between consecutive submissions. Figure 1 plots mean Δ pass rate as a function of submission index, stratified by feedback condition. We find that across all conditions, mean Δ pass rates fluctuate near zero, indicating that immediate improvement following feedback is neither consistent nor guaranteed. Positive gains at some indices are often followed by stagnation or regression at later attempts.

We observe modest early gains under natural language feedback, while test case feedback more frequently exhibits near-zero or negative changes. However, these patterns are not stable: trajectories overlap substantially, and no condition shows uniformly superior short-term improvement. Variability increases at later submission indices, which correspond to a shrinking subset of students who continue submitting after repeated failures—suggesting that feedback effects diminish and students may be attempting randomly at that stage.

4.2. Baseline Post-Feedback Performance

We summarize post-feedback performance using Δ pass rate, computed over valid submission transitions and excluding final submissions with no subsequent attempt. Table 1 reports descriptive statistics by condition. We find that mean Δ pass rates are near zero across all groups, and fewer than 35% of transitions result in positive gains. The large standard deviation reflects substantial heterogeneity in short-term responses.

Differences across conditions are modest: natural language feedback yields the highest probability of improvement, while test case feedback shows a slightly higher mean Δ and more submissions per student–problem on average. We fit a clustered logistic regression predicting improvement ($\Delta > 0$) with standard errors clustered at the student level, and find no significant differences across conditions (Wald test, $p = 0.50$). These patterns motivate our subsequent time-to-success analyses rather than serving as evidence of differential effectiveness at baseline.

Group	**Mean Δ**	**Std. Dev.**	**Pr($\Delta > 0$)**	**Avg. Sub**
No AI	0.020	0.459	0.314	1.75
Test Case	0.027	0.453	0.327	2.22
Natural Lang	0.017	0.449	0.345	1.66

Table 1: Baseline post-feedback performance by condition.

4.3. Effects of AI Feedback

Table 2: Associations between AI feedback and success and time to correctness

Comparison	**Prob. of Success**		**Time to Correctness**	
	Coef. (SE)	p	HR (95% CI)	p
NL vs. No AI	0.84 (0.30)	0.004	1.09 [1.01, 1.19]	0.03
TC vs. No AI	0.20 (0.24)	0.40	0.92 [0.85, 1.00]	0.05

We investigate whether AI-generated feedback is associated with students' ability to reach a correct solution and the effort required to do so. All analyses are at the student–problem level, including only initially incorrect submissions. We acknowledge that because feedback delivery is conditionally triggered rather than fully randomized, all results are associative rather than causal. Concretely, our estimand is the association between the condition *assigned to a failed submission* and subsequent outcomes—not a causal receipt effect.

Probability of Eventual Success For each student–problem pair, we define a binary outcome indicating whether the student ever reached a fully correct submission. We fit separate logistic regression models comparing each condition to the no-AI baseline with standard errors clustered at the student level. As shown in Table 2, we find that natural language feedback is significantly associated with higher eventual success ($\beta = 0.84$, $p = 0.004$), corresponding to an odds ratio of approximately 2.33: students receiving natural language feedback are more than twice as likely to eventually reach correctness. We find no significant association for test case feedback ($\beta = 0.20$, $p = 0.40$).

Time to Success We model time to correctness using Cox proportional hazards models with right censoring, where time is measured in submissions since first feedback exposure. We find that natural language feedback is associated with a modest but significant increase in the hazard of reaching correctness (HR = 1.09, $p = 0.03$): at any given attempt, students receiving natural language feedback are approximately 9% more likely to succeed than those receiving no feedback. Test case feedback, by contrast, shows a slightly lower hazard (HR = 0.92, $p = 0.05$), consistent with the longer submission sequences we observe descriptively. Together, our results suggest that natural language feedback improves both completion rates and convergence speed, while test case feedback offers no reliable advantage in either dimension.

4.4. Quality of Test Case Feedback

We categorize AI-generated test cases as **valid** (fails the student's submission but passes the reference solution), **false-positive** (fails both), or **no-effect** (exposes no discrepancy). We find that 66% of generated test cases are valid, 24% are false positives, and 10% have no effect. Submissions receiving valid test cases exhibit the highest average pass rates; false-positive and no-effect cases are associated with substantially lower performance. Importantly, we find that valid test cases are *not* associated with fewer subsequent submissions—higher-quality feedback improves correctness without shortening the debugging process. We believe this reflects the limited systematic debugging skills of introductory programming students, who struggle to efficiently exploit a counterexample even when it is correct.

4.5. Quality of Natural Language Feedback

To assess feedback quality, we evaluate 442 natural language feedback instances using GPT-4.1 as a judge, labeling each as *helpful*, *vague*, *incorrect*, or *harmful*. We find that 92.5% of instances are labeled helpful (409/442), 5.4% vague (24/442), 2.0% incorrect (9/442), and none harmful. Vague feedback typically offered general debugging suggestions without grounding them in the student's specific code; incorrect feedback reflected rare misattributions of the error source. Crucially, no feedback instance explicitly revealed a solution or bypassed student reasoning. Our results suggest that when quality failures occur, they are far more likely to involve insufficient specificity than pedagogically harmful guidance—supporting the safe deployment of natural language feedback in classroom settings.

5. Discussion

We find that natural language feedback is associated with both higher completion rates and faster convergence to correct solutions. We argue that this pattern is consistent with a cognitive load explanation: novice learners often struggle to interpret raw test failures, and natural language hints that translate program behavior into accessible explanations lower the barrier to identifying the next fix. Importantly, the significant hazard ratio in our survival analysis indicates that this effect reflects faster convergence, not merely a ceiling-driven success rate difference.

Test case feedback presents a more nuanced picture. We find its aggregate effect on outcomes to be weak, but we observe that this masks meaningful variation in feedback quality. Valid test cases are associated with higher pass rates, yet they do not reduce subsequent submission counts—suggesting that introductory students lack the systematic debugging skills needed to efficiently exploit a counterexample even when it correctly identifies a flaw. We believe this points to a fundamental mismatch between the form of the feedback and the skill level of the recipient, and motivates pairing test case feedback with explanatory guidance.

More broadly, our results highlight that evaluations of AI-assisted feedback should focus on learning trajectories rather than aggregate success rates, explicitly measure feedback quality, and interpret iteration counts cautiously—effective feedback may improve correctness without reducing submission effort. From a deployment standpoint, the test-case false-positive rate argues for governance safeguards: gating generated feedback—especially test cases—behind automated validity checks before it reaches students.

6. Limitations

We acknowledge several limitations of this work. First, although feedback conditions were randomized within labs, delivery was conditionally triggered by incorrect submissions, making all analyses associative rather than causal. Second, we do not include expert-authored human feedback as a baseline, so we cannot assess how AI-generated modalities compare to high-quality instructor feedback. Third, our outcome measures—test case pass rates, time to correctness, and iteration patterns—capture short-term dynamics rather than longer-term retention, conceptual understanding, or transfer to new problems. Fourth, feedback came from a single model (`gpt-4o-mini`) in one course at one institution with a mostly computer-science population, limiting generalizability to other models and populations. Fifth, feedback-quality labels were produced by an LLM judge (GPT-4.1), not human annotators, so we treat them as indicative pending human validation. Finally, the conditions are imbalanced (Group 0 largest), reducing precision for the smaller groups.

7. Future Work

Given these limitations, several directions remain open. One promising direction is developing models that predict test case invalidity before delivery, enabling real-time filtering or repair. Another is applying sequential and hierarchical models to characterize debugging pathways and persistence patterns across feedback conditions. More broadly, we hope ProgFeed can serve as a foundation for benchmarking AI feedback systems not only on

output quality, but on their downstream effects on student learning in authentic classroom settings.

8. Conclusion

We presented a large-scale classroom study and accompanying dataset, ProgFeed, examining how different forms of AI-generated feedback affect student performance in an introductory programming course. Across 6,693 submissions from 215 students, we find that natural language hints are significantly associated with higher completion rates and faster convergence to correct solutions, while test case feedback yields weak aggregate effects that depend critically on feedback validity. Notably, even valid test cases do not reduce subsequent submission counts, suggesting that introductory students lack the debugging skills to efficiently exploit counterexamples—pointing to a mismatch between feedback form and learner readiness. Our quality analyses further reveal that natural language feedback is overwhelmingly helpful (92.5%), while test case validity is far from guaranteed (66%), underscoring that evaluating feedback quality—not merely its presence—is essential. Together, these findings argue that the *form* of AI-generated feedback is not a superficial design choice but a determinant of pedagogical impact. We release ProgFeed to support further research on learning dynamics, feedback quality, and the responsible deployment of AI in educational settings.

References

Alireza Ahadi, Raymond Lister, Shahil Lal, and Arto Hellas. Learning programming, syntax errors and institution-specific factors. In *Proceedings of the 20th Australasian Computing Education Conference*, pages 99–108, 2018.

Neil CC Brown, Michael Kölling, Davin McCall, and Ian Utting. Blackbox: A large scale repository of novice programmers' activity. In *Proceedings of the 45th ACM Technical Symposium on Computer Science Education*, pages 223–228, 2014.

Hagit Gabbay and Anat Cohen. Combining LLM-generated and test-based feedback in a MOOC for programming. In *Proceedings of the Eleventh ACM Conference on Learning @ Scale (L@S)*, pages 177–187, 2024.

Kacy Johnston. The effects of immediate correctness feedback on student learning, understanding, and achievement, 2015. URL https://api.semanticscholar.org/CorpusID:188952447. Thesis.

Majeed Kazemitabaar, Justin Chow, Carl Ka To Ma, et al. Studying the effect of ai code generators on supporting novice learners in introductory programming. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*, pages 1–23, 2023.

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

Ekaterina Kochmar, Dung D. Vu, Robert Belfer, Varun Gupta, Iulian Serban, and Joelle Pineau. Automated personalized feedback improves learning gains in an intelligent

tutoring system. *Artificial Intelligence in Education*, 12164:140 – 146, 2020. URL https://api.semanticscholar.org/CorpusID:218516674.

Nischal Ashok Kumar and Andrew Lan. Improving socratic question generation using data augmentation and preference optimization, 2024a.

Nischal Ashok Kumar and Andrew Lan. Using large language models for student-code guided test case generation in computer science education, 2024b.

Juho Leinonen, Arto Hellas, Sami Sarsa, Brent Reeves, Paul Denny, James Prather, and Brett A Becker. Using large language models to enhance programming error messages. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1*, pages 563–569, 2023.

Benjamin Paaßen, Barbara Hammer, Thomas William Price, Tiffany Barnes, Sebastian Gross, and Niels Pinkwart. The continuous hint factory-providing hints in vast and sparsely populated edit distance spaces. *arXiv preprint arXiv:1708.06564*, 2017.

Maciej Pankiewicz and Ryan S Baker. Large language models (gpt) for automating feedback on programming assignments. *arXiv preprint arXiv:2307.00150*, 2023.

Tung Phung, José Cambronero, Sumit Gulwani, Tobias Kohn, Rupak Majumdar, Adish Singla, and Gustavo Soares. Generating high-precision feedback for programming syntax errors using large language models. *arXiv preprint arXiv:2302.04662*, 2023a.

Tung Phung, Victor-Alexandru Pădurean, José Cambronero, Sumit Gulwani, Tobias Kohn, Rupak Majumdar, Adish Singla, and Gustavo Soares. Generative ai for programming education: Benchmarking chatgpt, gpt-4, and human tutors. In *Proceedings of the 2023 ACM Conference on International Computing Education Research (ICER), Volume 2*, pages 41–65, 2023b.

Tung Phung, Victor-Alexandru Pădurean, Anjali Singh, Christopher Brooks, José Cambronero, Sumit Gulwani, Adish Singla, and Gustavo Soares. Automating human tutor-style programming feedback: Leveraging gpt-4 tutor model for hint generation and gpt-3.5 student model for hint validation. *arXiv preprint arXiv:2310.03780*, 2023c.

Chris Piech, Jonathan Huang, Andy Nguyen, Mike Phulsuksombati, Mehran Sahami, and Leonidas Guibas. Learning program embeddings to propagate feedback on student code. In *International conference on machine Learning*, pages 1093–1102. PMLR, 2015.

Ruchir Puri, Dennis S Kung, Geert Janssen, Wei Zhang, et al. Project codenet: A large-scale ai for code dataset for learning a diversity of coding tasks. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, 2021.

Michihiro Yasunaga and Percy Liang. Graph-based, self-supervised program repair from diagnostic feedback. In *International Conference on Machine Learning*, pages 10799–10808. PMLR, 2020.

Michihiro Yasunaga and Percy Liang. Break-it-fix-it: Unsupervised learning for program repair. In *International Conference on Machine Learning*, pages 11941–11952. PMLR, 2021.