Scaling Formative Feedback in Programming Education through Mastery Learning and AI

Mr. Weihang Zhang, Duke University

Weihang Zhang is a Master of Engineering student in Electrical and Computer Engineering at Duke University, with a concentration in Machine Learning and Artificial Intelligence. His research interests include large language models for education, automated formative feedback systems, and scalable software infrastructure for programming assessment.

Dr. Javier Pastorino, Duke University

Dr. Javier Pastorino is an Assistant Professor in Electrical and Computer Engineering at Duke University. His work bridges academia and industry, drawing on more than twenty years of experience across roles in software engineering. He holds a Ph.D. in Computer Science with a focus on artificial intelligence, and his research centers on data management, machine learning for scientific applications, and privacy-aware machine learning from a software engineering perspective. In addition to his research contributions, he is actively engaged in enhancing engineering education through the development of scalable, technology-driven learning systems.

Scaling Formative Feedback in Programming Education through Mastery Learning and AI

Weihang Zhang, Javier Pastorino
weihang.zhang@duke.edu, javier.pastorino@duke.edu
Pierre R. Lamond Department of Electrical and Computer Engineering
Duke University

Abstract. The mastery learning framework emphasizes formative assessment, where students are given opportunities to fail, receive feedback, and improve until they reach proficiency. In this approach, timely and consistent feedback is essential, yet remains a major challenge in large-enrollment programming courses. Current mastery learning platforms typically rely on automated testing, which provides correctness but offers little guidance on code quality or style. As a result, students may achieve functional solutions without receiving formative insights until instructional staff intervene, creating a resource bottleneck.

This pilot study explores the integration of large language models (LLMs) into an existing mastery learning platform for programming courses. Our system operates in three phases: (1) automated correctness testing through traditional test cases, (2) generation of targeted semantic feedback via a secure, institutionally hosted LLM API once a correctness threshold is achieved, and (3) post-processing to refine the feedback so that it remains constructive without over-directing the student's problem-solving process. The system enables instructors to customize the assessment criteria, aligning feedback with specific learning objectives.

We evaluated the system by deploying it in a graduate-level programming course in the machine learning track at Duke University, using a within-subject, two-stage survey design. Baseline surveys captured student experiences with grade-only feedback, while post-deployment surveys assessed perceived clarity, usefulness, efficiency, and motivational impact of the LLM-augmented feedback. Although this pilot study received feedback from a small cohort (N = 9), the qualitative results indicated that students preferred the AI-supported feedback over the grade-only baseline as they reported improved interpretability, reduced error diagnosis time, and lower perceived reliance on teaching assistants.

This work provides evidence that workflow-constrained LLM pipelines, rather than raw model capability, deliver scalable, pedagogically aligned formative feedback. The findings suggest a practical hybrid assessment model where deterministic grading ensures correctness, LLMs provide structured interpretive guidance, and instructors focus on higher-order conceptual learning.

Keywords: Programming education, formative feedback, mastery learning, large language models, rubric-aligned assessment

1. Introduction

High-quality feedback is widely recognized as a central component of effective programming education, supporting students in identifying errors, refining their solutions, and developing durable conceptual understanding [1], [2]. However, delivering timely and meaningful feedback remains a persistent challenge, particularly in programming courses with growing enrollments [3]. As class sizes increase, instructors and teaching assistants face substantial workload constraints, often limiting feedback to correctness judgments or brief comments that provide little insight into students' underlying misconceptions [4].

To address scalability concerns, automated grading systems have become a standard component of programming courses. These systems enable rapid and consistent evaluation of student submissions, allowing learners to iterate through resubmissions and enabling instructors to manage large cohorts efficiently. While effective for checking functional correctness, such tools typically provide limited explanatory or formative feedback, leaving many students uncertain about why their solutions fail or how to improve beyond satisfying test cases [3], [5]. This trade-off between scalability and pedagogical depth has spurred research into automated feedback mechanisms that support learning without adding instructional burden.

Recent advances in large language models (LLMs) have renewed interest in this problem by demonstrating an unprecedented capacity to process source code and natural language jointly. Unlike rule-based systems, LLMs can generate natural-language explanations, contextualize errors, and adapt feedback to individual student submissions. Early studies suggest that students often perceive LLM-generated programming feedback as helpful and supportive [6], and that such feedback can approximate aspects of instructor commentary in readability and structure [7]. These findings position LLMs as a promising means of augmenting existing assessment workflows.

However, directly deploying LLMs for programming feedback presents new challenges. Technically, unguided models may hallucinate non-existent errors or fail to align with course-specific rubrics [8]. Pedagogically, current applications often prioritize evaluation efficiency (e.g., automating grading [9]) or error correction (e.g., fixing bugs [6]), while overlooking the critical phase of solution refinement. Overemphasizing the direct generation of solutions may inadvertently foster passive learning and diminish the exploratory process required for skill acquisition [10]. Consequently, there is a growing need for structured approaches that strictly align LLM capabilities with formative learning goals while maintaining consistency with established assessment practices.

One particularly relevant pedagogical framework for addressing this need is mastery learning, first formalized by Bloom, which emphasizes that students are expected to achieve a predefined level of mastery for each instructional unit, supported by formative assessment and corrective instruction, before advancing to subsequent topics [11]. Unlike traditional time-based instruction, mastery learning treats achievement as the primary constant while allowing time and instructional support to vary, enabling learners to iteratively refine their understanding through cycles of assessment, feedback, and revision [12]. Central to this framework is the use of frequent formative assessment, coupled with targeted corrective feedback, to diagnose and address misconceptions, ensuring that learning gaps are resolved before advancement [13], [14]. This iterative, feedback-

driven process aligns naturally with programming education, where students must repeatedly debug, revise, and refine their solutions to achieve conceptual understanding. However, providing such individualized and iterative feedback at scale remains a significant instructional challenge [3].

Motivated by this gap, we present the Automated Delivery, Testing, and Grading (ADTG)-LLM system, a rubric-aware feedback pipeline designed to complement, rather than replace, traditional autograders. Instead of treating the LLM as a standalone evaluator, our approach embeds it within a role-based workflow that strictly decouples technical diagnosis from pedagogical feedback generation. This hybrid architecture enforces alignment with assignment-specific rubrics and incorporates automated guardrails against solution leakage. The system was deployed in a graduate programming course, *Programming and Data Structures for Machine Learning* (P4ML), and integrated seamlessly into the existing GitLab-based submission infrastructure via a secure, institutionally-hosted API.

The contributions of this work are threefold. First, we propose a multi-stage pipeline that decouples diagnosis from feedback synthesis to prioritize conceptual refinement over simple error correction, ensuring strictly rubric-aligned guidance without fine-tuning the underlying model. Second, we demonstrate the practical feasibility of layering LLM-generated feedback atop existing deterministic autograders using standard enterprise APIs and workflow engineering, without incurring the computational burden of training proprietary models. Finally, we report findings from a pilot classroom deployment, suggesting that the proposed system reduces debugging overhead, lowers perceived reliance on instructional staff, and improves students' overall feedback experience compared to a grade-only baseline. Together, these contributions demonstrate how LLMs can be systematically integrated into engineering education settings to support scalable, mastery-oriented formative feedback. The proposed framework is designed for seamless adoption within existing course infrastructures, enabling instructors to deliver high-quality feedback without increasing instructional workload, particularly in large-enrollment programming courses.

2. Related work

Automated assessment of programming assignments has a long history in computing education, dating back to early systems that executed student programs against predefined test cases to determine correctness [15]. Over time, automated grading tools have evolved to incorporate static analysis and more sophisticated test-case-based frameworks, enabling scalable evaluation and consistent judgment across large cohorts [5]. These systems substantially reduce instructor workload and support repeated resubmissions, making them widely adopted in programming courses [5].

However, despite these advantages, the feedback provided by autograders is predominantly diagnostic rather than developmental. Tools typically report whether a submission passes or fails a set of test cases while highlighting mismatched outputs or syntactic issues, but they offer little guidance towards conceptual understanding, explanations of errors, or strategies for improvement [3]. This limitation becomes especially evident in large-enrollment courses, where students who repeatedly fail test cases often struggle to identify underlying misconceptions without meaningful explanatory feedback [4]. Moreover, these approaches rarely evaluate higher-level code quali-

ties, such as readability, maintainability, documentation quality, or students' reasoning processes [5]. Adoption of these systems can also present challenges for instructors. For example, they may need to learn tool-specific specification schemas, such as JSON- or JUnit-based templates, which introduces an additional onboarding burden and constrains pedagogical flexibility [16]. Although autograders provide efficient and scalable correctness checking, they struggle to deliver formative, explanatory feedback that helps students interpret errors, refine their solutions, and build durable programming competencies. Their limitations indicate the need for approaches capable of supporting deeper reasoning and more meaningful feedback in programming education.

Recent advances in large language models (LLMs) have introduced new opportunities for providing richer feedback in programming education. Unlike previous generations of educational tools that relied on predefined rules or unit tests, LLMs possess the capability to process source code and natural language semantically, enabling nuanced interpretation of student intent [17]. Empirical studies have confirmed the utility of this capability, showing that GPT-based systems can produce personalized hints that students generally rate as useful and supportive for iterative improvement [6]. Specifically, Dai et al. [7] found that LLM-generated feedback often matches or exceeds human feedback in terms of readability and coverage of formative feedback components, suggesting that LLMs can serve as a promising complement to existing assessment workflows. Building on this potential, further research demonstrates LLMs' intrinsic capacity to generate feedback with high specificity and clarity that aligns with formative assessment standards [18]. Moreover, these models exhibit the pedagogical versatility to produce distinct feedback structures, such as corrective, elaborative, or metacognitive prompts, thereby offering adaptable support suited for structured assessment workflows [19].

Despite the promise of LLM-assisted feedback, existing approaches exhibit several limitations that hinder reliable classroom deployment. A primary concern is the tendency of these models to prioritize solution generation over instructional scaffolding. Studies indicate that LLMs often provide direct code fixes rather than guiding hints, which can inadvertently encourage passive learning and deprive students of the cognitive struggle necessary for skill acquisition [10], [20]. Second, regarding reliability, LLM-generated feedback can be fluent yet unfaithful to the student artifact, with hallucinated or unsupported claims that reduce trust and complicate instructional use [8]. Third, aligning output with specific course expectations remains difficult; generic prompting often fails to adhere to rubric-based guidance, resulting in feedback that lacks submission specificity and usefulness for targeted debugging [8], [21]. Finally, controlling the pedagogical form of feedback is non-trivial. While LLMs can generate different feedback styles, recent work suggests they exhibit significant variability and often fail to consistently produce the intended structure without sophisticated constraints [19], [22]. Together, these findings highlight the need for structured, course-aligned pipelines to ensure faithfulness, pedagogical appropriateness, and consistency.

These technical limitations help explain why the recent adoption of LLMs in engineering education has been cautious, primarily limiting their role to automated evaluation where human oversight is feasible. Consistent with this pattern, recent frameworks like FlexiGrader [9] and StatGrad [23] leverage LLMs to scale the grading of open-ended or handwritten solutions. Even when studies move beyond simple scoring, Auby et al. [24] use LLMs to analyze students' short-answer explanations and identify reasoning patterns; however, the primary utility remains framed

around enhancing the evaluator's insight into student performance. Moreover, where direct student feedback is explored, it predominantly focuses on error correction—generating hints to help students debug broken code to satisfy test cases [6].

However, this focus on error resolution carries pedagogical risks. Recent work has investigated students' use of ChatGPT to generate or correct programming solutions, raising concerns that such interactions may reduce productive struggle and shift students toward passive consumption of answers rather than active reasoning [25]. Functional correctness is only the first step toward mastery; a significant gap remains in supporting the refinement of solutions without replacing the student's cognitive effort.

By positioning the LLM as a post-submission mentor that triggers only after baseline correctness is met, we shift the focus from "fixing bugs" to "cultivating engineering quality," ensuring that AI support serves to deepen, rather than bypass, conceptual understanding.

3. Methodology

3.1 Pedagogical objectives

In order to address the limitations of traditional autograding systems, the ADTG-LLM system was designed and guided by three core objectives. First, the feedback was designed to be formative rather than merely diagnostic, with the goal of actively supporting students in refining and optimizing their code and in bridging the gap between basic functional correctness and full conceptual mastery. Second, all generated feedback was required to be pedagogically aligned with the assignment-specific rubric, ensuring that every comment was directly grounded in explicit learning objectives and remained consistent with instructional intent. Finally, the system was engineered for scalability, integrating seamlessly into the existing submission workflow so that high-quality, individualized feedback could be delivered automatically and immediately, without imposing additional administrative or grading burden on teaching staff.

3.2 System architecture and integration

The feedback infrastructure is built on a Kubernetes-based microservices architecture that interfaces directly with the course's GitLab repositories. This repository-based submission model enables students to submit assignments by pushing code to their designated repositories. When a student submits a grading request, a dedicated grader process is triggered to clone the latest submission and execute the deterministic autograder. By decoupling persistent storage from compute and execution, the system ensures that grading remains both isolated from user environments and scalable across concurrent submissions.

Built upon this scalable infrastructure, the system was architected to augment, rather than replace, the existing autograding infrastructure through a competency-gated workflow. The grading process starts by running the deterministic autograder to verify functional correctness. If the submission score falls below the passing threshold required to unlock subsequent assignments (70% in our course), the system returns the standard autograder output, including test-case results and basic diagnostic information, allowing students to revise and resubmit their solutions.

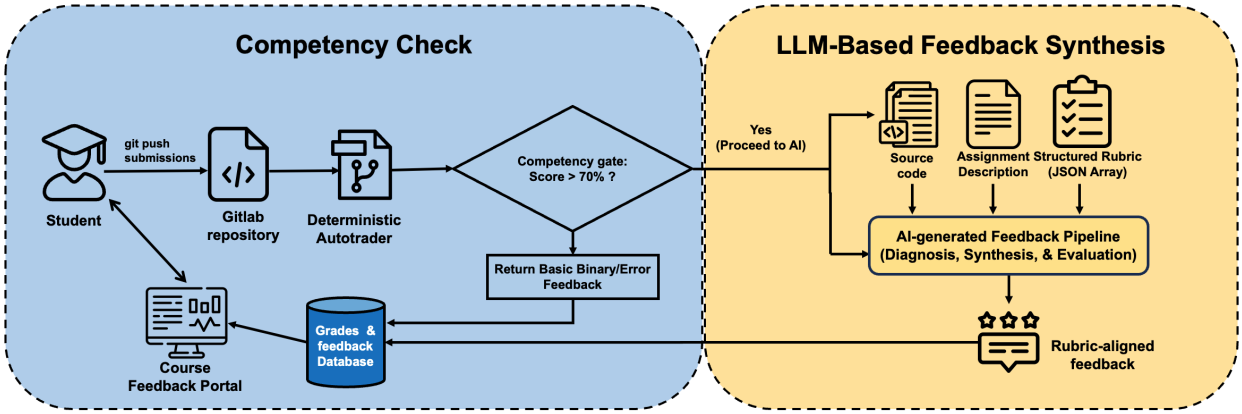

Figure 1: **Overview of the competency-gated feedback pipeline.**

When the submission score exceeds the passing threshold, the LLM-based feedback pipeline is executed by aggregating the student's source code, the assignment description, and the assignment-specific structured rubric into a unified input context. The resulting context is then provided to the pipeline to generate structured, rubric-aligned AI feedback, as depicted in Figure 1. This hybrid approach ensures that students address basic implementation errors independently, while leveraging the LLM to provide targeted guidance for optimization, refinement, and achieving full mastery of the assessment materials.

3.2.1 Rubric alignment and context injection

A key pedagogical requirement was that all feedback, whether human- or machine-generated, must align strictly with the assessment criteria to ensure consistency with instructional goals. To this end, each assignment was accompanied by a comprehensive rubric specifying targeted learning objectives, including *algorithmic efficiency*, *data structure selection*, and *code modularity*. This rubric served not merely as a grading tool but as a roadmap to identify specific areas for improvement, thereby enhancing the student's learning experience beyond simple correctness.

To operationally enforce this alignment, each assignment is governed by a hierarchical evaluation rubric stored in a standardized JSON format rather than unstructured text. Each rubric item is encapsulated as an object containing a `"title"` (e.g., "Naming and Convention") and a list of specific `"descriptions"` (e.g., "Follow PEP8", "Use docstrings"). Instead of flattening this data, the system injects the raw JSON array directly into the LLM's system prompt. This design leverages the model's native capability to parse structured data, forcing the LLM to explicitly map its feedback to specific JSON keys, enabling traceability to defined course objectives.

3.2.2 Data privacy and safety

Data privacy was a central design constraint governing the system's implementation. All LLM interactions were routed through a secure, university-hosted enterprise API endpoint established in partnership with the model provider. Unlike public-facing services, this infrastructure ensures that student data is strictly contained, effectively preventing submissions from being stored on public servers or used to train general foundation models.

3.2.3 Model specification and configuration

To ensure technical reproducibility and feasibility, the system utilizes a large language model (GPT-4.1) accessed via an institutionally hosted enterprise API endpoint. This infrastructure is provided through a partnership between the host institution and the model provider (Microsoft Azure OpenAI), ensuring that all interactions are governed by strict data privacy agreements compliant with FERPA regulations. Crucially, this study does not involve pre-training or fine-tuning a foundational model, which addresses the scalability concerns regarding computational resources. Instead, the system relies on the pre-trained model's reasoning capabilities, guided by the rubric-aware prompting pipeline described in the *AI-Generated Feedback Pipeline* section. Temperature settings were dynamically adjusted across stages: a lower temperature ($T = 0.1$) was used for the *Technical Diagnosis* stage to maximize deterministic behavior, while a moderate temperature ($T = 0.7$) was applied during *Feedback Synthesis* to encourage naturalistic pedagogical tone.

3.2.4 AI-generated feedback pipeline

The feedback is generated by a sequence of three stages, namely the *technical diagnosis*, the *parallel feedback synthesis*, and the *selection and deployment*, as depicted in Figure 2.

The first stage forces the LLM to act in a strictly analytical capacity and instructs it to act as a rubric-based auditor, explicitly avoiding pedagogical commentary. The model audits the student submission against the preprocessed, structured rubric on a criterion-by-criterion basis.

Each rubric `"title"` is treated as an independent evaluation dimension, for which the model produces a binary pass/fail judgment accompanied by a concise, evidence-based justification grounded in observable properties of the source code (e.g., *"Naming and Convention: FAILED-variable names are non-descriptive and violate PEP8 guidelines"*). By isolating rubric-based defect detection from instructional feedback generation, the system produces a stable intermediate *Technical Diagnosis Report*, a grounded intermediate representation that informs all subsequent stages and reduces the risk of hallucinated or spurious issues during feedback generation.

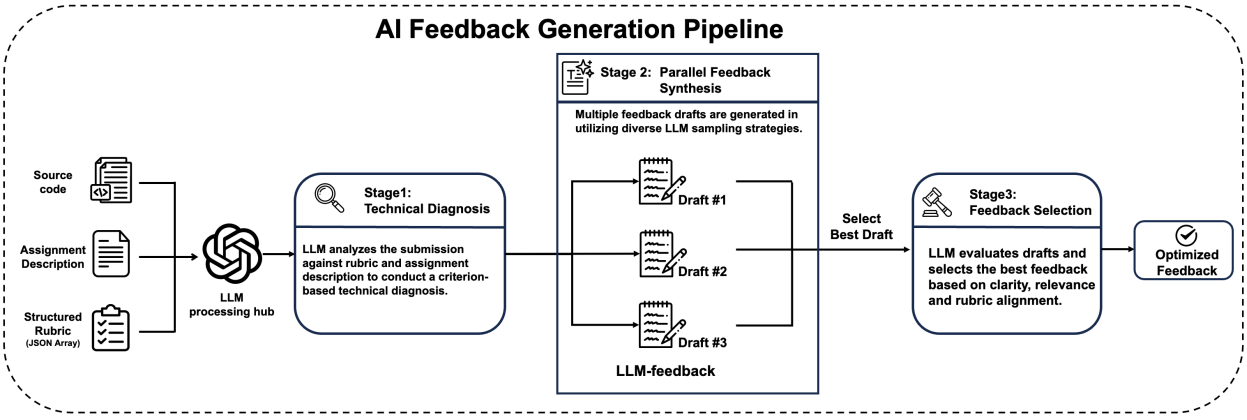

Figure 2: **Overview of the LLM-based feedback generation pipeline.**

In the second stage, the *parallel feedback synthesis*, the system proceeds to formative feedback generation using the technical diagnosis report as grounding evidence. To mitigate stochastic variability inherent in LLM outputs, the pipeline employs a parallel generation strategy with $n = 3$ independent candidate drafts. In this stage, the model is instructed to generate formative feedback in a teaching-oriented style, emphasizing clarity, encouragement, and alignment with instructional objectives. Each response follows a diagnosis-grounded structure: it explicitly references satisfied rubric criteria, explains identified deficiencies by connecting them to specific rubric dimensions, and offers abstracted guidance strategies aimed at improvement without disclosing full solutions.

Finally, in the *selection and deployment* stage, the system applies a supervisory selection mechanism in which the model evaluates all three candidate drafts and selects the optimal response based on clarity, tone, and consistency with the Technical Diagnosis Report. Further, a strict academic integrity guardrail serves as a normalizing mechanism to prevent solution leakage by immediately rejecting any candidate draft when explicit code blocks or direct fixes are detected. Instead of attempting partial edits, the system discards the compliance-violating output and triggers a full regeneration cycle, up to a maximum of $k = 3$ attempts, prompting the model to generate a fresh response adhering to the abstraction constraints. This post-processing step acts as a quality assurance gate to preserve the instructional intent and decrease the likelihood of students receiving feedback that does not adhere to the structural and pedagogical standards defined.The finalized feedback is then published to the course feedback portal alongside standard autograder results.

3.3 Study design

To assess the efficacy of the proposed ADTG-LLM system, we conducted a comparative pilot study within a live classroom environment. This section details the study context, participant demographics, survey instruments, and procedural workflow used to capture student perceptions.

3.3.1 Context and participants

The study was conducted during Fall 2025 in the context of *P4ML: Programming and Data Structures for Machine Learning*, a graduate-level course designed to bridge foundational programming concepts with machine learning workflows. The curriculum featured a series of programming assignments targeting both algorithmic correctness and conceptual understanding. The course enrolled more than 50 students, of whom approximately 98% were international students.

The study protocol was approved by the Institutional Review Board (IRB) of Duke University (Protocol #2026-0099). Prior to deployment, all participants provided written informed consent. To ensure privacy, strict anonymity was maintained, and participation was voluntary with no impact on course grades. Ultimately, a total of nine students (N=9) completed the full sequence of feedback interactions and corresponding surveys.

Limitation Note: *The focus of this pilot is exploratory in nature, and given the limited sample size, our primary objective is to identify preliminary usage patterns and qualitative trends rather than to establish statistically generalizable effects across a broader population.*

3.3.2 Experimental procedure

We employed a within-subject design to minimize confounding variables such as student ability or assignment difficulty. In this setup, all participants were exposed to two distinct feedback paradigms. Under the **Control Condition (Grade-Only)**, students received standard output from the deterministic autograder, restricted to numerical scores and basic pass/fail status while having access to teaching assistants for help. Conversely, in the **Experimental Condition (ADTG-LLM)**, participants interacted with our proposed pipeline, which generated multi-stage, rubric-aware feedback.

Data was collected using two surveys administered after assignment completion:

- **Survey A (Baseline Experience):** Assessed students' interactions with the Grade-Only condition. Key dimensions included the sufficiency of numerical scores, the time required to diagnose errors, and the frequency of help-seeking behaviors (e.g., consulting TAs).
- **Survey B (System Effectiveness):** Evaluated the ADTG-LLM pipeline. Items focused on feedback clarity, perceived utility for debugging, impact on code quality, and comparative satisfaction against the baseline.

Both surveys utilized a standardized 5-point Likert scale (1 = Strongly Disagree; 3 = Neutral; 5 = Strongly Agree) to quantify student perceptions, alongside multi-select behavioral questions and open-ended text fields for qualitative thematic analysis.

4. Results

This section details the findings from the pilot deployment, contrasting the limitations of the traditional workflow with the utility of the ADTG-LLM system.

4.1 Limitations of Grade-Only feedback

Survey A (baseline) revealed substantial limitations in the traditional grade-only autograding workflow. Instead of exhibiting a clear consensus, student perceptions of grade-only feedback were highly fragmented, indicating that numerical scores alone failed to provide reliable explanatory value. More importantly, downstream behavioral indicators revealed a significant cognitive and operational burden associated with this feedback format.

As summarized in Table 1, over half of the students reported spending more than 30 minutes just **attempting to interpret** the meaning of their grade and the grader pass/fail cases, while one-third of respondents reported making more than ten independent debugging attempts when faced with failed test cases.

Results also showed that students needed additional support from the instructional team to understand the test cases results; specifically, most students sought help from teaching assistants while also engaging in peer discussions. Less than half of respondents reported reaching out to the lead instructor for additional help.

Qualitative responses from the survey further reinforced these quantitative trends. Across open-ended questions, students most frequently reported a lack of diagnostic clarity, repeatedly stat-

Item	Percentage
Time spent interpreting grade $>$ 30 minutes	55.6%
More than 10 self-debugging attempts	33.3%
Asked a TA for clarification	77.8%
Discussed the grade with classmates	66.7%
Asked the instructor for clarification	44.4%
Accepted grade without inquiry	22.2%

Table 1: **Distribution of student experiences under grade-only feedback** ($N = 9$)

ing that they *"didn't know where the code was wrong"* or struggled with *"understanding why I got this wrong"*. Consistent with the quantitative findings, these responses indicate that while grade-only feedback scales efficiently for scoring, it provides limited support for debugging and conceptual understanding.

It is worth noting that such diagnostic struggle is not inherently unproductive. Effortful debugging can support deeper learning when students have sufficient scaffolding to engage meaningfully with the challenge [26]. However, the qualitative responses suggested that students were primarily unable to locate the source of errors rather than reasoning about underlying concepts, a pattern more consistent with diagnostic uncertainty than with productive cognitive struggle.

4.2 Utility and perception of ADTG-LLM feedback

Participants generally responded positively to the ADTG-LLM system, with results indicating that the system significantly reduced the cognitive load required to diagnose errors. Time-on-task patterns suggest a marked improvement in interpretability compared to the baseline. Over half of the respondents (55.6%) reported spending less than 15 minutes reviewing and applying the LLM-generated feedback, a sharp contrast to the grade-only condition where a majority spent over 30 minutes just analyzing the test failures to fix their code.

These efficiency gains generally align with perceptions of clarity, though student experiences varied. As summarized in Table 2, over half of the cohort (55.6%) rated the ADTG-LLM feedback as *Clear* or *Very Clear*. However, a non-trivial subset of students (22.2%) reported the feedback as *Unclear*, with an additional 22.2% expressing neutral views, indicating variability in perceived clarity across participants. This polarization, where roughly one-quarter of students struggled with clarity, correlates with qualitative comments, where students explicitly noted that high-level pedagogical advice was occasionally interpreted as vague when they sought immediate, granular fixes.

Despite these variations in perceived clarity, the system successfully delivered value through its multi-dimensional design. When asked to identify the most valuable aspects of the feedback, students offered a balanced distribution of preferences (Figure 3). The identification of specific errors (33.3%) was the most cited benefit, followed equally by explanations of concepts, style suggestions, and efficiency tips. This distribution suggests that the system provided a holistic assessment rather than focusing exclusively on a single aspect, such as syntax.

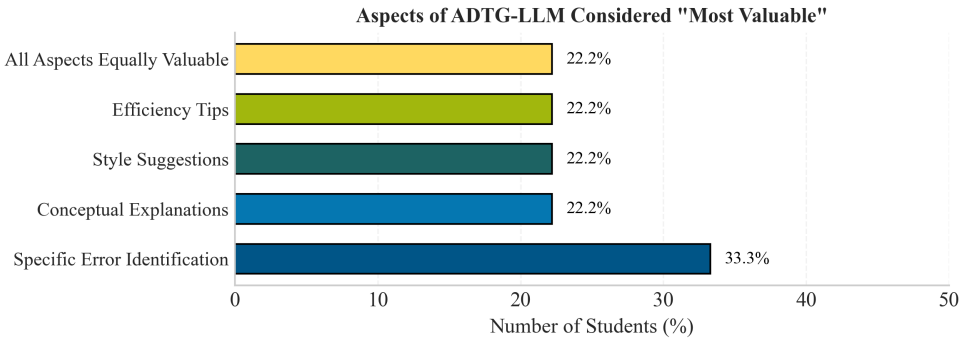

Figure 3: **Distribution of Perceived Utility.**

Ultimately, this feedback translated into measurable improvements in student work. As detailed in Table 2, most students reported a moderate to significant improvement in their code quality after reviewing the ADTG-LLM feedback. At the same time, the engagement behaviors are polarized, with the same proportion of students reporting using the feedback to fix most issues as those reporting reading the feedback without implementing immediate changes. This dichotomy suggests a clear strategic engagement pattern. While students understood the advice (given the quantitative data), they likely exercised discretion, prioritizing critical error repairs over the stylistic or optimization suggestions previously identified as valuable but non-blocking.

Metric / Response Category	**Count**	**Percentage**
1. Impact on Code Quality		
Significant improvement	2	22.2%
Moderate improvement	5	55.6%
Minimal / No impact	2	22.2%
(Total Positive Impact)	*(7)*	*(77.8%)*
2. Feedback Clarity		
Very Clear	2	22.2%
Clear	3	33.3%
Neutral	2	22.2%
Unclear / Very Unclear	2	22.2%
(Total Positive Clarity)	*(5)*	*(55.6%)*
3. Usage Behavior		
Fixed most or all issues	1	11.1%
Fixed some issues	3	33.3%
Read but made no changes	4	44.4%
Ignored feedback	1	11.1%

Table 2: **Student Engagement, Perception, and Outcomes** $(N = 9)$

4.3 Comparative advantage

Direct comparative questions further demonstrate student preference for the proposed system. As can be observed in Table 3, a majority of respondents (66.7%) rated the LLM feedback as better or much better, while the remaining respondents viewed it as approximately equivalent. Notably, no respondent rated the LLM feedback as worse than the baseline.

Furthermore, more than half of the respondents agreed that the ADTG-LLM feedback reduced the time they would otherwise spend seeking help from teaching assistants, while two-thirds affirmed that it was more effective than the baseline at supporting their understanding of code strengths and weaknesses. Most students identified that ADTG-LLM would be a useful tool for future students as it would enhance their performance in the course and support their learning.

Comparative Metric / Statement	**% Agree or Strongly Agree**
Perceived Feedback Quality of ADTG-LLM vs. Grade-Only	
Feedback was Better or Much-Better	66.7%
Feedback was About the Same	33.3%
Feedback was Worse or Much-Worse	—
Effectiveness vs. Baseline	
Supported better understanding of strengths/weaknesses	66.7%
Reduced time spent seeking help (TA/Instructor)	55.6%
Future Outlook	
Valuable for use in future assignments	66.7%

Table 3: **Student Perception of ADTG-LLM** ($N = 9$)

5. Discussion

The results of this pilot study provide preliminary evidence that a structured, rubric-aware LLM pipeline can effectively bridge the feedback deficit in automated programming assessment. By layering interpretive guidance atop deterministic grading, the ADTG-LLM system addressed the twin challenges of scalability and pedagogical depth.

5.1 Efficacy of the separation of concerns architecture

Unlike generic chatbot interfaces or simple wrapper tools, a critical finding of this work is that high-quality feedback emerged not from raw LLM capabilities, but from the architectural constraints imposed by our multi-stage pipeline. Prior literature warns that unguided LLMs frequently hallucinate errors or leak solutions [10]. Our results suggest that the multi-stage separation of concerns effectively mitigated these risks.

Qualitative observations during the preliminary design phase highlighted the necessity of our multi-stage approach. As illustrated in Table 4, direct zero-shot prompting exhibited significant instability. It frequently hallucinated non-existent errors (e.g., criticizing correct formatting) and

generated unstructured text that failed to meet the required JSON schema. Furthermore, direct prompting often bypassed pedagogical goals by explicitly leaking code solutions.

Direct Zero-Shot LLM

Output: Unstructured Text with Hallucinations

```
Your code fails.  You have extra spaces on line 5 that violate PEP8.
Here is the fixed code:
def factorial(n):
  if n == 0:  return 1
    return n * factorial(n-1)
```

[Problem 1] Hallucinates non-existent formatting errors were present.

[Problem 2] Directly reveals the solution.

ADTG-LLM Pipeline

Output: Structured JSON (Rubric-Aligned)

```
{
  "summary":  {
    "star":  3.5,
    "title":  "Summary",
    "description":  "Logic error detected in recursive step..."
  },
  "feedback":  [
    {
      "star":  2,
      "title":  "Recursion Logic",
      "description":  "Recursive call uses n instead of n-1, causing infinite recursion."
    },
    {
      "star":  5,
      "title":  "Formatting",
      "description":  "Good adherence to PEP8.  No spacing issues found."
    }
  ]
}
```

[Advantage] Structured, factual, rubric-grounded, and non-leaking.

Table 4: **Qualitative Case Study: Hallucination vs. Structured Reliability.**

In contrast, the ADTG-LLM pipeline produced strictly formatted, rubric-aligned JSON payloads. By leveraging the diagnostic validation in Stage I, the system mitigated hallucinations regarding code style, while the safety checks in Stage III successfully prevented solution leakage. This suggests that reliability in educational AI depends on workflow engineering: strictly constraining the model to a chain of verification ensures that feedback remains factual, structured, and formative.

5.2 Implications for instructional scalability

From an instructional perspective, the system demonstrates how AI can serve as a scalable first line of defense. The observed reduction in students' perceived reliance on teaching assistants (given the significant reduction in clarification-seeking) suggests that structured LLM feedback can alleviate the clarification burden commonly associated with large programming courses.

Moreover, this outcome was achieved without replacing existing infrastructure but instead layering LLM feedback on top of deterministic grading. This hybrid approach preserved the rigorous correctness checking of the autograder while augmenting it with interpretive guidance. This aligns with recent calls in the literature for AI systems that complement, rather than replace, established assessment workflows, allowing human staff to focus on complex mentoring rather than routine debugging.

Furthermore, by utilizing a standard, institutionally hosted API rather than a custom-trained model, this system demonstrates high technical feasibility. It avoids the resource-intensive burden of model training, proving that effective educational AI can be achieved through workflow engineering on top of off-the-shelf models.

5.3 Reaffirming feedback as a learning tool

The strong student preference for written explanations and conceptual clarification underscores the pedagogical value of feedback that goes beyond correctness. Our results indicate that the ADTG-LLM system successfully shifted the user experience from evaluative (getting a grade) to formative (learning from errors). The fact that a majority of students actively used the suggestions to revise their code challenges the notion that AI tools inevitably lead to passive consumption.

Moreover, the reduction in time-on-task observed under the ADTG-LLM condition should not be interpreted as the elimination of productive struggle. As suggested by our results, the time spent under the grade-only condition appears to be associated more with diagnostic search than with higher-order conceptual reasoning. By providing structured interpretive guidance, the system may help redirect student effort toward more meaningful engagement with code quality and design principles. Whether this redirection leads to improved learning outcomes remains an open question for future work.

By enforcing strict academic integrity guardrails that strip out direct solutions, the system supported scaffolded learning, requiring students to engage in the cognitive process of repair and reinforcing the role of feedback as a dynamic learning tool.

5.4 Fairness and feedback consistency

An important design consideration was ensuring consistency and fairness in feedback quality across submissions. By grounding all feedback in a shared, rubric-based diagnostic report and enforcing structured output constraints via the post-processing sanitization layer, the system reduces variance arising from prompt sensitivity or stylistic drift. This post-processing step acts as a normalizing filter, ensuring that all students receive feedback adhering to the same structural and pedagogical standards, regardless of stochastic variations in the model's raw output. However, we do not claim that the system eliminates all forms of bias, and future work will include auditing feedback consistency across student demographics and assignment types.

6. Conclusion and future work

This work presents ADTG-LLM, a hybrid assessment framework designed to reconcile the long-standing trade-offs between instructional scalability and pedagogical depth in programming education. By layering a structured, rubric-aware LLM pipeline on top of traditional deterministic autograding, the system bridges the gap between binary correctness checks and actionable formative guidance, without disrupting existing assessment workflows.

Our qualitative observations illustrate that by strictly decoupling technical diagnosis, feedback synthesis, and post-processing integrity checks, the system can effectively mitigate common LLM risks (e.g., hallucinations and solution leakage) without resource-intensive model fine-tuning. This suggests that effective educational AI relies not only on the capabilities of the underlying model, but also on the rigorous workflow engineering that constrains it.

From a pedagogical perspective, the system shifted the student experience from passive grade consumption toward active, scaffolded debugging and reflection. Preliminary evidence indicates that rubric-aligned, explanatory feedback can reduce students' reliance on instructional staff for clarification while supporting greater independence in diagnosing and resolving errors. Importantly, these benefits were achieved by augmenting instead of replacing the deterministic autograder, preserving its role as an objective arbiter of functional correctness.

While promising, these findings must be interpreted within the context of the exploratory pilot study. Participants self-selected into the study, potentially introducing a bias where more motivated students were overrepresented. Moreover, the small sample size and the single-course setting limited the generalization of the findings. However, the qualitative feedback obtained suggests that this work is moving in the right direction and further validates the system's architecture and its capacity to transform raw diagnostics into high-quality pedagogical feedback, enabling scaling up to larger class cohorts.

This work lays the foundation for a scalable, human-in-the-loop assessment paradigm that integrates automated feedback with instructional oversight. More broadly, the core design principles underlying ADTG-LLM are not specific to the course in which the system was piloted. Its modular architecture and use of institutionally hosted API endpoints, rather than custom-trained models, further lower adoption barriers, allowing the framework to be integrated into other engineering courses that rely on iterative, competency-based assessment with minimal overhead. This positions the framework as a practical starting point for programs seeking to deliver pedagogically meaningful feedback at scale.

Future work will explore different avenues to enhance the system and this exploratory study. Due to resource constraints, we restricted the scope of feedback generation to submissions that met the passing threshold for the assessments. This decision may have inadvertently excluded struggling students, therefore, we will explore adaptive thresholds designed specifically for novices grappling with fundamental errors. We will also enhance the system's instrumentation to support real-time self-evaluation of generated feedback, enabling early identification of at-risk students while timely intervention remains possible. In addition, we plan to deploy the system to more diverse contexts to evaluate learning outcomes and feedback fairness and consistency.

References

[1] J. Hattie and H. Timperley, "The power of feedback," *Review of educational research*, vol. 77, no. 1, pp. 81–112, 2007.

[2] D. J. Nicol and D. Macfarlane-Dick, "Formative assessment and self-regulated learning: A model and seven principles of good feedback practice," *Studies in higher education*, vol. 31, no. 2, pp. 199–218, 2006.

[3] H. Keuning, J. Jeuring, and B. Heeren, "A systematic literature review of automated feedback generation for programming exercises," *ACM Transactions on Computing Education*, vol. 19, 1 Sep. 2018, ISSN: 19466226. DOI: 10.1145/3231711.

[4] G. Haldeman, A. Tjang, M. Babeş-Vroman, *et al.*, "Providing meaningful feedback for autograding of programming assignments," in *SIGCSE 2018 - Proceedings of the 49th ACM Technical Symposium on Computer Science Education*, vol. 2018-January, Association for Computing Machinery, Inc, Feb. 2018, pp. 278–283, ISBN: 9781450351034. DOI: 10.1145/3159450.3159502.

[5] M. Messer, N. C. Brown, M. Kölling, and M. Shi, "Automated grading and feedback tools for programming education: A systematic review," *ACM Transactions on Computing Education*, vol. 24, 1 Feb. 2024, ISSN: 19466226. DOI: 10.1145/3636515.

[6] M. Pankiewicz and R. S. Baker, "Large language models (gpt) for automating feedback on programming assignments," *arXiv preprint arXiv:2307.00150*, 2023.

[7] W. Dai, Y.-S. Tsai, J. Lin, *et al.*, "Assessing the proficiency of large language models in automatic feedback generation: An evaluation study," *Computers and Education: Artificial Intelligence*, vol. 7, p. 100 299, 2024.

[8] Q. Jia, J. Cui, H. Du, *et al.*, "Llm-generated feedback in real classes and beyond: Perspectives from students and instructors," in *Proceedings of the 17th International Conference on Educational Data Mining*, 2024, pp. 862–867.

[9] A. Frias, S. Krishnakumar, and A. Pandey, "Flexigrader: An llm-based personalized autograder to enable flexible and open-ended creative exploration in cs1," in *2025 ASEE Annual Conference & Exposition*, Montreal, Quebec, Canada: ASEE Conferences, Jun. 2025. DOI: 10.18260/1-2--56580. [Online]. Available: https://peer.asee.org/56580.

[10] J. Prather, B. N. Reeves, P. Denny, *et al.*, ""it's weird that it knows what i want": Usability and interactions with copilot for novice programmers," *ACM transactions on computer-human interaction*, vol. 31, no. 1, pp. 1–31, 2023.

[11] B. S. Bloom, "Learning for mastery. instruction and curriculum. regional education laboratory for the carolinas and virginia, topical papers and reprints, number 1.," *Evaluation comment*, vol. 1, no. 2, n2, 1968.

[12] M. Winget and A. M. Persky, "A practical review of mastery learning," *American journal of pharmaceutical education*, vol. 86, no. 10, ajpe8906, 2022.

[13] P. Black and D. Wiliam, "Assessment and classroom learning," *Assessment in Education: principles, policy & practice*, vol. 5, no. 1, pp. 7–74, 1998.

[14] T. R. Guskey, "Closing achievement gaps: Revisiting benjamin s. bloom's "learning for mastery"," *Journal of advanced academics*, vol. 19, no. 1, pp. 8–31, 2007.

[15] J. C. Paiva, J. P. Leal, and Á. Figueira, "Automated assessment in computer science education: A state-of-the-art review," *ACM Transactions on Computing Education*, vol. 22, 3 Jun. 2022, ISSN: 19466226. DOI: 10.1145/3513140.

[16] A. Agrawal and B. Reed, "A survey on grading format of automated grading tools for programming assignments," in *Proceedings of the 15th International Conference of Education, Research and Innovation (ICERI)*, ser. ICERI2022, arXiv:2212.01714, Seville, Spain: IATED, Nov. 2022, pp. 7506–7514, ISBN: 978-84-09-45476-1. DOI: 10.21125/iceri.2022.1912. [Online]. Available: https://doi.org/10.21125/iceri.2022.1912.

[17] P. Denny, J. Prather, B. A. Becker, *et al.*, "Computing education in the era of generative ai," *Communications of the ACM*, vol. 67, no. 2, pp. 56–67, 2024.

[18] Z. Zhang, Z. Dong, Y. Shi, T. Price, N. Matsuda, and D. Xu, "Students' perceptions and preferences of generative artificial intelligence feedback for programming," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 38, 2024, pp. 23 250–23 258.

[19] D. Lohr, H. Keuning, and N. Kiesler, "You're (not) my type-can llms generate feedback of specific types for introductory programming tasks?" *Journal of Computer Assisted Learning*, vol. 41, no. 1, e13107, 2025.

[20] N. Kiesler and D. Schiffner, "Large language models in introductory programming education: Chatgpt's performance and implications for assessments," *arXiv preprint arXiv:2308.08572*, 2023.

[21] A. Pathak, R. Gandhi, V. Uttam, *et al.*, "Rubric is all you need: Enhancing llm-based code evaluation with question-specific rubrics," *arXiv preprint arXiv:2503.23989*, 2025.

[22] T. A. N. Frederick Eneye, C. F. Ijezue, A. Imam Amjad, M. Amjad, S. Butt, and G. Castañeda-Garza, "Advances in auto-grading with large language models: A cross-disciplinary survey," in *Proceedings of the 20th Workshop on Innovative Use of NLP for Building Educational Applications (BEA 2025)*, E. Kochmar, B. Alhafni, M. Bexte, *et al.*, Eds., Vienna, Austria: Association for Computational Linguistics, Jul. 2025, pp. 477–498, ISBN: 979-8-89176-270-1. DOI: 10.18653/v1/2025.bea-1.35. [Online]. Available: https://aclanthology.org/2025.bea-1.35/.

[23] A. M. PEng, M. Talebi-Kalaleh, M. Sabek, *et al.*, "Automated grading of engineering mechanics assignments using large language models and computer vision: A work in progress," in *2025 ASEE Annual Conference & Exposition*, Montreal, Quebec, Canada: ASEE Conferences, Jun. 2025. DOI: 10.18260/1-2--55492. [Online]. Available: https://peer.asee.org/55492.

[24] H. Auby, N. Shivagunde, A. Rumshisky, and M. Koretsky, "Using machine learning to analyze short-answer responses to conceptually challenging chemical engineering thermodynamics questions," 2024. DOI: 10.18260/1-2--48236. [Online]. Available: https://par.nsf.gov/biblio/10538630.

[25] L. Guertault, "Investigating the capabilities and limitations of chatgpt to perform programming assignments from an introductory r programming course," in *2025 ASEE Annual Conference & Exposition*, Montreal, Quebec, Canada: ASEE Conferences, Jun. 2025. DOI: 10.18260/1-2--56894. [Online]. Available: https://peer.asee.org/56894.

[26] E. L. Bjork, R. A. Bjork, *et al.*, "Making things hard on yourself, but in a good way: Creating desirable difficulties to enhance learning," *Psychology and the real world: Essays illustrating fundamental contributions to society*, vol. 2, no. 59-68, pp. 56–64, 2011.