Toward Cross-Domain Automated Feedback: A Comparative Evaluation of Open-Source Models across Diverse Student Assessment Types

Muhammad Haseeb[1], Min Paing Hmue[1], Ahmad Imam Amjad[2], Maaz Amjad[1], Victor S. Sheng[1]

[1]Texas Tech University, Texas, USA, [2]The University of Punjab, Pakistan,
mhaseeb@ttu.edu, mhmue@ttu.edu, ahmadimamamjad@gmail.com, maaz.amjad@ttu.edu, victor.sheng@ttu.edu

Abstract

Constructive, personalized, and timely feedback is essential to student learning. However, providing such feedback in large classes remains a major challenge. Large language models (LLMs) offer alternatives to support instructors by generating relevant feedback that highlights both student strengths and areas for improvement. Nevertheless, most existing LLM-based feedback systems rely on proprietary APIs and are often tailored to specific tasks, which limits their accessibility, flexibility, and applicability in resource-constrained educational settings. In this study, we investigate the potential of two open-source LLMs (DeepSeek R1 and Qwen 3.5) to support automated feedback generation across three disciplines (e.g., programming assignments, essays, and mathematics problems). We evaluate two prompting strategies (unified and multi-agent) across these domains and use human judgment criteria to assess feedback quality. Through this analysis, we examine the potential of open-source models as practical, scalable alternatives to proprietary API-based systems for developing freely accessible feedback-generation tools. Our results show that a single open-source model can generate useful feedback across diverse domains, though with varying effectiveness. DeepSeek R1 performs better on reasoning-intensive tasks such as mathematics, while Qwen 3.5 works best for holistic tasks such as writing, but both models struggle with programming tasks. We find that model architecture matters more than prompting strategy, and reasoning-optimized models excel in structured domains, while general-purpose models perform better on holistic tasks. Finally, we conclude that a multi-agent approach does not consistently guarantee improved results over a single unified LLM approach.

1 Introduction

Feedback plays an important role in the learning process (Hattie and Timperley, 2007). Constructive and timely feedback helps students identify gaps in their understanding, refine their skills, and improve academic performance (Cavalcanti et al., 2021). However, the usefulness of feedback in educational settings depends not only on its quality but also on how quickly the feedback is given. Students report significantly lower motivation when feedback is delayed and takes more than 10 days to arrive (Fisher et al., 2025). In contrast, personalized and timely feedback can improve academic performance and overall learning outcomes (Aucejo and Wong, 2025). However, instructors find it difficult to provide timely, high-quality, and individualized feedback in large classes and resource-constrained educational settings due to workload constraints (Messer et al., 2024). This leads to delayed feedback, inconsistencies in grading, poor feedback quality, and tracking individual student progress over time becomes more challenging (Haseeb et al., 2026). To address these challenges, artificial intelligence (AI) and large language models (LLMs) offer opportunities to complement instructor support by enabling automated feedback generation and addressing scalability issues.

LLM-based systems have been proposed to automate the process of feedback generation (Murtazin et al., 2025; Haseeb et al., 2026). These systems help reduce feedback turnaround time from days to seconds (Haseeb et al., 2026), and mitigate delays associated with manual grading and feedback delivery (Dong et al., 2024; Guizani et al., 2025; Kinder et al., 2025; Nkoyo et al., 2025). Recent studies demonstrate that LLM-generated feedback can support students' learning goals, enhance motivation, and improve engagement, especially when the feedback is adaptive and responsive to learners' needs (Murtazin et al., 2025; Kinder et al., 2025), which makes them well-suited for both traditional classroom and online learning environments (Cisneros-González et al., 2025).

Most state-of-the-art feedback generation sys-

Proceedings of the 21st Workshop on Innovative Use of NLP for Building Educational Applications, pages 951–963

tems used in AI-assisted educational applications rely on proprietary models (e.g., GPT-4, Gemini, and Claude). However, dependence on these models creates transparency, reproducibility, and accessibility-related issues. These limitations are important because they hinder the practical deployment of such tools in resource-constrained academic institutions, where reliance on commercial AI systems or the computational infrastructure required to deploy large-scale models may not be practical or sustainable. Therefore, it is important to explore whether open-source models can serve as a reasonable alternative that addresses these limitations for automated feedback generation.

Open-source language models provide freely accessible alternatives for educators and practitioners. In educational settings, they are particularly valuable because they can support transparency, reproducibility, adaptability, local deployment, and cost control (Lin et al., 2026). In addition, open-source models have shown promising results in targeted feedback-generation tasks. For example, open-source models have been found to perform nearly on par with proprietary models in generating and evaluating feedback in programming education (Koutcheme et al., 2025). However, prior research has largely examined these models in isolated educational tasks and within narrow disciplinary contexts. Most of the existing literature is concentrated on a single-domain setup, which includes computer science-related settings (e.g., programming education (Koutcheme et al., 2025; Cisneros-González et al., 2025)), writing assessment (Alnemrat et al., 2025; Gilmore, 2025), and mathematical reasoning tasks (Lee et al., 2026; Rigaud Téllez et al., 2024). However, relying on separate tools for different assessment types is often impractical in real educational settings. Consequently, it remains unclear whether a single open-source model can generate useful feedback across diverse assessment types and disciplines (e.g., programming, essay writing, and mathematics). This gap motivates the need for a broader cross-disciplinary evaluation of open-source language models for feedback generation.

To the best of our knowledge, no previous study has examined a single LLM-based model (as a unified framework) for feedback generation across three disciplines: (i) mathematics, (ii) academic writing, and (iii) programming. To address this gap, we extend previous work on LLM-based feedback generation (Haseeb et al., 2026) to examine LLM-based feedback generation across mathematics, academic writing, and programming within a single unified framework.

Research Question: To what extent can a single task-specialized open-source model generate useful and pedagogically meaningful feedback across different assessment types while maintaining consistent feedback quality? For example, can a coding-specialized model also provide useful feedback on essay-based assignments and mathematics problems? We further hypothesize that decomposing feedback generation into a multi-agent setup can yield deeper and more domain-specific feedback insights than a unified single prompt approach.

The main contributions of this paper are as follows:

1. We extend LLM-based feedback generation beyond programming to two additional disciplines: (i) mathematics, and (ii) academic writing.

2. We collect and curate datasets for each of the three disciplines, and then examine 2 open-source models across 3 domains, to assess feedback quality.

2 Methodology

We evaluate LLM-based feedback generation using two prompting approaches applied to two open-source models across three educational domains: (i) programming, (ii) academic writing, and (iii) mathematics. Figure 1 illustrates the overall framework. In the **Unified Approach**, each model independently receives the full context, including the problem, student submission, and evaluation criteria, in a single prompt and produces feedback in one inference call.

In the **Multi-Agent Approach**, feedback generation is decomposed into specialized subtasks assigned to distinct agents, each focused on a specific aspect of evaluation, followed by a synthesis step where applicable. Both approaches are evaluated across programming, academic writing, and mathematics to assess whether prompting architecture affects feedback quality across domains.

2.1 Datasets

To evaluate LLM feedback generation across educational domains, we curate datasets from three distinct disciplines: programming, academic writing, and mathematics. Each dataset contains student

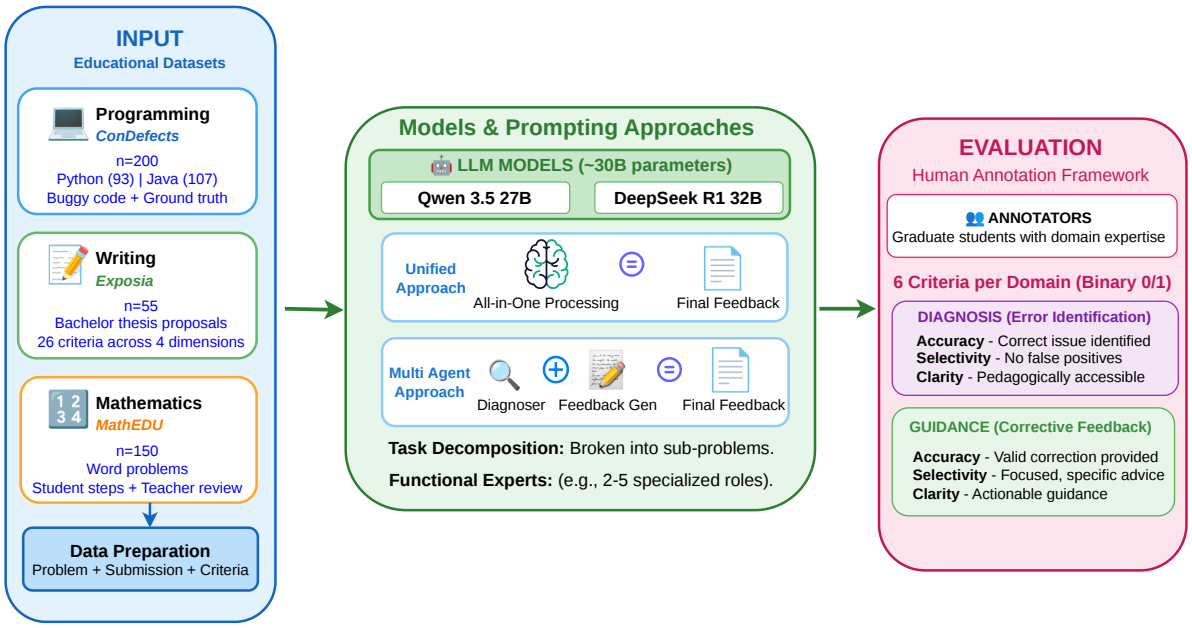

Figure 1: Framework for LLM-based feedback generation across domains with unified and multi-agent prompting.

submissions with errors, along with their associated ground truth. This enables systematic evaluation of feedback quality. Table 1 summarizes the key characteristics of each dataset.

Dataset	Samples	Languages	Source
ConDefects	200	Python (93) Java (107)	(Wu et al., 2023)
Exposia	55	English	(Zyska et al., 2026)
MathEDU	150	English	(Hsu et al., 2026)

Table 1: Datasets Overview

2.1.1 Programming: ConDefects

We use ConDefects dataset (Wu et al., 2023) for evaluating the feedback in the programming task. It consists of real buggy code submissions from competitive programming contests on AtCoder. This dataset contains student errors made during the competition, which reflect students' understanding of the code, misconceptions, and mistakes.

We use 200 students' submissions across two programming languages: 93 submissions written in Python, and 107 submissions in Java. Each sample includes four elements: (i) a problem description, (ii) the student's buggy code, (iii) the corrected version, and (iv) the exact line where the student made a mistake. These problems are of intermediate difficulty level, and each student submission in this set exhibits distinct programming concepts, error types, and logical errors. Since we did not have access to the reference feedback, we rely on human assessment to evaluate the quality of LLM feedback. The evaluation assesses whether the LLM accurately identifies the bug while providing pedagogically useful and personalized guidance.

2.1.2 Writing: Exposia

For the academic writing domain, we use the Exposia dataset (Zyska et al., 2026), which comprises 55 bachelor's thesis research proposals from TU Darmstadt. Each proposal represents a student's initial draft and the final proposal after incorporating the teacher's and fellow students' feedback. Each proposal includes a problem statement, a literature review, a methodology, and a writing quality assessment.

The dataset also includes a well-defined scoring criterion that teachers and fellow student reviewers use to assess submission quality. For our evaluation, we use teacher feedback and assigned scores as references to guide human evaluators, but we do not treat them as absolute ground truth, since assessing academic writing quality is inherently subjective. Therefore, human evaluators utilize their own judgment to make final quality assessments, informed by but not bound to instructor scores.

We utilize the 26 criteria across four dimensions provided with the dataset (Zyska et al., 2026), originally used by teachers to assess submissions and generate feedback. We incorporate these criteria directly into our prompts to guide LLM-based feedback generation.

Problem & Research Question (MO1-MO7, 9 points): Evaluates if there is an engaging hook, a clear problem statement, a domain specification, a relevant problem, a discussion of current approaches, a properly formulated research question, and scope feasibility.

Literature & Theory (AP1-AP7, BI1-BI2, 13 points): Evaluates state-of-the-art coverage, relevance demonstration, weakness identification, differentiation from prior work, synthesis quality, and theoretical framework quality.

Methodology (AP8, MD1-MD7, 8 points): Examines methodology availability, completeness, relevance, reference to existing materials, discussion of difficulties, limitations acknowledgment, and implementation details.

Writing Quality (LA1-LA2, 4 points): Evaluates the language quality (grammar, clarity, terminology) and structural coherence (logical flow, consistent language, and common thread).

Each criterion uses a defined rubric with scores ranging from 0-1 or 0-2 points, yielding a maximum of 33 points. We use the expert annotations (type: "draft", role: "expert") as ground truth for evaluating LLM scoring accuracy.

2.1.3 Mathematics: MathEDU

For the mathematics domain, we use 150 problems from the MathEDU dataset (Hsu et al., 2026), which is designed to evaluate mathematical reasoning and tutoring capabilities. Each sample consists of a word problem, the student's incorrect answer, and their solution process, which shows the students' reasoning steps. The dataset contains diverse mathematical concepts, such as ratio calculations, arithmetic operations, distance-time-speed problems, probability, permutations, and geometric reasoning. Each sample includes a *teacher_review* field containing expert feedback on the student's error, and provides a reference ground truth for evaluating the quality of LLM-generated feedback.

2.2 Open-source Models

We select two state-of-the-art open-source large language models, each model representing a distinct architectural paradigm: (i) a general-purpose model, and (ii) a reasoning-oriented model. Both models operate at a comparable scale (approx 30B parameters). This ensures that any observed performance differences reflect model design and reasoning capabilities rather than the parameter count.

1. **Qwen 3.5 27B** (Yang et al., 2025) is a general-purpose model achieving state-of-the-art performance on standard benchmarks. We use the BF16 precision variant with thinking mode disabled via Ollama, generating responses directly without explicit reasoning traces. This setup reflects standard deployment in efficiency-constrained environments for educational feedback.

2. **DeepSeek R1 32B** (Guo et al., 2025) is a reasoning-distilled model. It compresses chain-of-thought capabilities from larger models into a smaller architecture. The model generates explicit thinking traces before producing the final response. We use the FP16 precision variant with default settings.

Our model selection balances three considerations. First, comparable parameters ensure a fair comparison. Second, both models are top performers within their paradigm at this scale. Third, the 30B parameter range allows deployment on a single-GPU infrastructure. This reflects practical deployment constraints, supports replicability, and improves accessibility in low-resource educational environments. This pairing helps examine whether explicit model reasoning improves feedback quality or whether general-purpose generation that follows the rubrics suffices to maintain feedback quality in educational domains.

2.3 Prompting Approaches

We evaluate two distinct prompting approaches for generating educational feedback in three disciplines (programming, essay writing, and mathematics): (i) a unified single-prompt approach, and (ii) a multi-agent prompting approach. These approaches are compared in terms of the quality of feedback generated between both models.

2.3.1 Unified Approach

The unified approach generates feedback through a single LLM call. The model receives the full context problem, student submission, and evaluation criteria, and produces a single feedback response.

For the programming task (see Figure 2), the prompt includes the problem description, related test cases, the programming language, and the student's buggy code. It also includes instructions to identify the error and provide feedback on a conceptual misunderstanding. The output is 150-250

words, which focuses on bug identification and pedagogical guidance. For the writing task (see Figure 3), the prompt includes the research proposal and 26 scoring criteria organized across four dimensions. The model produces scores in a standardized format. It then gives 150-250 words of feedback to highlight key issues and suggest improvements. For the mathematics task (see Figure 4), the prompt includes the problem, the student's answer, and their solution process. The model identifies incorrect steps, diagnoses underlying misconceptions, and provides guidance to address these gaps. This approach mirrors how educators review student work and assess all aspects of their work at once. Moreover, this approach is computationally efficient, requiring only one inference call per student submission.

2.3.2 Multi-Agent Approach

The multi-agent approach divides feedback generation into specialized sub-tasks. Each agent is assigned a specific evaluation or feedback generation step, and, if necessary, a synthesis step to produce coherent and comprehensive feedback. Each agent is implemented as a separate LLM call, where the output of one stage serves as input to the next.

In the programming domain (see Figure 5–6), two agents are utilized:

- Error Analyzer: This agent produces a structured diagnosis that includes the location of the error, the type of error, the root cause, and the misconception of the student and the direction of the correction.
- Feedback Generator: This agent receives the diagnosis, the problem statement, and the submission, and crafts pedagogically helpful feedback for the student.

In the writing domain, four dimension-specific agents are employed in conjunction with a synthesizer:

- Problem and RQ Agent: Scores the problem and research question and provides 2-3 sentence feedback on clarity and relevance.
- Literature Agent: Evaluates the quality of the literature review section and provides 2-3 sentences on quality, coverage, and synthesis.
- Methodology Agent: Evaluates completeness and relevance of methodology and provides feedback on quality, completeness, and rigor.
- Writing Quality Agent: Scores language quality, consistency, and structure while providing feedback.
- Synthesizer: Integrates all scores and analyses to formulate a unified feedback response of 150 to 250 words for the student.

In the mathematics domain, two agents are utilized:

- Error Analyzer: Outputs error location along with error type and conceptual gap, with a small description of the mistake.
- Feedback Generator: Receives diagnosis with problem description and submission, and formulates personalized feedback for students in 50-120 words.

We include representative prompts for the programming domain in the appendix (Figures 2, 5, 6). Prompts for writing and mathematics follow the same structure and are publicly available in the accompanying repository, along with sample model outputs for all domains[1].

2.3.3 Design Rationale

Our architectures are motivated by domain-specific task-decomposition principles from prior work, in which complex evaluation tasks are decomposed into distinct role-based cognitive sub-tasks (Wang et al., 2025). This persona-based decomposition improves the analytical depth and feedback quality.

Programming. We adopt a two-stage pipeline (diagnosis $\rightarrow$ feedback) following code review frameworks (Yang et al., 2024). Debugging in programming first requires identifying the precise source, location, and the type of error, whereas the corrective guidance needs to provide pedagogically meaningful feedback. Separating bug identification helps the respective agent to focus more on the required task. This decomposition addresses the limitations of LLMs in generating correct code without accurately localizing conceptual errors.

Writing. We decompose our evaluation strategy by rubric dimensions, which aligns with writing assessment practices where content, structure, and language require distinct evaluative expertise (Hashemi et al., 2024). The four-agent structure maps directly to our selected dataset rubric dimensions. Finally, a synthesizer integrates these

[1] https://github.com/haseeb2001/crossdomainfeedback

dimension-specific analyses into coherent feedback, resolving potential conflicts and avoiding fragmented responses.

Mathematics. Our pipeline mirrors cognitive tutoring approaches, in which diagnosing misconceptions precedes corrective instruction (Liu et al., 2023). In mathematical problems, errors are often rooted in conceptual misunderstandings or omissions during the solving process, rather than surface-level mistakes; therefore, step-by-step error diagnosis enables targeted, personalized remediation rather than generic solution guidance.

The multi-agent paradigm draws from recent work demonstrating that task decomposition improves LLM performance on complex reasoning tasks (Wang et al., 2025). We hypothesize that specialized agents can provide deeper, domain-specific, and more precise feedback than a unified approach, especially for tasks like mathematics that require pedagogical reasoning.

2.4 Experimental Setup

All experiments were conducted on NVIDIA H100 GPUs (96GB VRAM) using Ollama (v0.6.8) for local inference. Qwen 3.5 27B was executed on a single GPU, whereas DeepSeek R1 32B required two GPUs due to higher memory requirements. For both models, we used deterministic decoding with a temperature set to 0 and a maximum output length of 4096 tokens. All other generation parameters were kept at their default values. These settings were applied consistently across all experiments. All our prompts and inference scripts are publicly available [2].

2.5 Evaluation Framework

We evaluate generated feedback through human annotation using six criteria per domain, organized into two dimensions: *diagnosis* (error identification) and *guidance* (corrective feedback).

Each dimension is assessed along three aspects: **Accuracy**—correctly identifies the underlying issue and/or provides a valid correction; **Selectivity**—avoids false positives and focuses only on relevant issues; **Clarity**—pedagogically accessible and appropriate for the student's level.

For programming, we adopt evaluation criteria from (Koutcheme et al., 2025). For writing and mathematics, we developed parallel criteria to enable consistent cross-domain comparison under a unified evaluation framework. The full set of evaluation criteria is provided in Table 2.

Domain	Code	Definition
Programming (Explanation / Fix)		
	EA	Bug description matches ground truth
	ES	No false positives identified
	EC	Explanation understandable to a novice
	FA	Fix would resolve the bug
	FS	No extraneous modifications
	FC	Clear and usable code suggestions
Writing (Issue / Suggestion)		
	WA	Writing issue correctly identified
	WS	No overcriticism of acceptable choices
	WC	Understandable to student writer
	GA	Suggestion would improve writing
	GS	Focused and specific advice
	GC	Actionable with concrete guidance
Mathematics (Error / Solution)		
	MA	Mathematical error correctly identified
	MS	Valid alternative paths not penalized
	MC	Appropriate for student level
	SA	Guidance leads to correct solution
	SS	Addresses specific error only
	SC	Step-by-step actionable guidance

Table 2: Evaluation criteria across domains. All criteria scored binary (0/1).

Annotation Protocol. Evaluation was conducted by four annotators for programming and mathematics, and three annotators for writing, proportional to dataset size. All annotators were graduate students with relevant domain expertise. Each criterion was scored binary (0 = No, 1 = Yes). Final scores were aggregated by averaging across annotators for each sample and then across all samples. Inter-annotator agreement was measured using Cohen's κ on overlapping samples, which indicates moderate agreement ($\kappa = 0.48$) across configurations.

3 Results

This section presents evaluation results across three educational domains: programming (Section 3.1), writing (Section 3.2), and mathematics (Section 3.3). All results are reported using the six-criterion evaluation framework introduced in Section 2.5 (see Table 2 for criterion definitions).

3.1 Programming Results

Programming feedback presents the greatest challenge across all three domains, with average scores (0.27-0.48), which is substantially lower than mathematics (0.70-0.91) and writing (0.35-0.81). Table 3 presents the complete results.

[2] https://github.com/haseeb2001/crossdomainfeedback

DeepSeek R1 outperforms Qwen 3.5 across all conditions (0.43-0.48 vs 0.27-0.29), with the unified approach yielding marginally better results than the multi-agent approach for both models. A striking pattern emerges in criterion-level scores: explanation clarity (EC) approaches perfection (0.82-1.00) across all configurations, whereas explanation accuracy (EA: 0.12-0.20) and fix accuracy (FA: 0.09-0.55) remain comparatively low. This difference shows that models can produce fluent, well-articulated feedback but often fail to correctly identify underlying bugs and conceptual errors, rendering their feedback ineffective. This pattern of fluent feedback that fails to identify actual bugs confirms prior observations on programming feedback (Haseeb et al., 2026).

Configuration	EA	ES	EC	FA	FS	FC	Avg
Qwen 3.5-U	0.17	0.10	1.00	0.14	0.15	0.20	0.29
Qwen 3.5-MA	0.12	0.06	0.97	0.09	0.13	0.27	0.27
DeepSeek R1-U	0.20	0.18	0.86	0.55	0.55	0.52	**0.48**
DeepSeek R1-MA	0.14	0.15	0.82	0.54	0.54	0.40	0.43

Table 3: Programming domain results (n=200). U = Unified, MA = Multi-Agent.

3.2 Writing Results

Qwen 3.5 substantially outperforms DeepSeek R1 across all conditions, achieving an average score of 0.81 compared to 0.35–0.38 for DeepSeek. This performance gap is more pronounced in accuracy metrics: Qwen achieves 0.91–0.93 on both issue identification (WA) and suggestion accuracy (GA), while DeepSeek scores only 0.16–0.18 on these criteria. This pattern indicates that DeepSeek frequently fails to identify correct writing issues and produces ineffective suggestions.

Both models exhibit high selectivity (WS: 0.91–0.95), indicating they avoid unnecessary criticism. However, for DeepSeek, this occurs alongside extremely low accuracy, giving the impression of a permissive model that avoids criticism by missing issues entirely. This suggests that the model is selectively incorrect rather than effectively discriminative, highlighting that high selectivity does not translate to high-quality feedback when underlying issues remain undetected.

Further differences emerge in suggestion quality. While Qwen maintains moderate clarity and actionability (GC: 0.78), its suggestion specificity (GS) remains relatively low (0.49–0.58), indicating that even when models identify writing shortcomings, they struggle to produce highly targeted guidance. DeepSeek shows a similar pattern, performing poorly across all suggestion dimensions (GS: 0.04, GC: 0.29–0.36), reflecting a lack of focused and actionable feedback.

The choice of prompting approach shows minimal impact for both models. Qwen achieves identical average scores (0.81) with both approaches, while DeepSeek shows marginal variation (0.35–0.38). This suggests that for writing feedback, model capability is a stronger determinant of quality than prompting strategy. Unlike structured domains, writing evaluation appears to be a holistic task that does not benefit substantially from decomposition into specialized subtasks. Table 4 presents evaluation results for the writing domain across 55 research proposals.

Configuration	WA	WS	WC	GA	GS	GC	Avg
Qwen 3.5-U	0.93	0.91	0.82	0.91	0.49	0.78	**0.81**
Qwen 3.5-MA	0.91	0.93	0.75	0.91	0.58	0.78	**0.81**
DeepSeek R1-U	0.16	0.95	0.56	0.18	0.04	0.36	0.38
DeepSeek R1-MA	0.18	0.93	0.51	0.18	0.04	0.29	0.35

Table 4: Writing domain results (n=55). U = Unified, MA = Multi-Agent.

3.3 Mathematics Results

DeepSeek R1 with the unified approach achieves the highest overall performance (0.91), outperforming all other configurations. Both models exhibit near-perfect error detection (MA, MS $\approx$ 1.0), indicating that identifying mathematical mistakes is comparatively feasible, as models can follow a step-by-step approach and mathematical steps are more grounded in rules, suggesting that dealing with this level of mathematical mistakes is not a limiting factor for current LLMs.

The main difference lies in the guidance provided in the solution. DeepSeek with unified settings achieves substantially higher solution accuracy (SA: 0.81) than Qwen-Unified (0.25) and both multi-agent configurations (0.03–0.11). This pattern highlights that DeepSeek's reasoning capabilities are best utilized in unified settings to generate mathematically valid corrective guidance.

It is also observed that, unlike programming and writing, these models are highly sensitive to prompting approaches for mathematics. The multi-agent approach significantly degrades performance for both models, with average scores dropping from 0.79 to 0.75 for Qwen and from 0.91 to 0.70 for

DeepSeek. The degradation is more pronounced in clarity (MC) and solution accuracy (SA), indicating that task decomposition may fragment the coherent reasoning required for effective mathematics feedback. Table 5 presents the evaluation results for the mathematics domain across 150 problems.

Configuration	MA	MS	MC	SA	SS	SC	Avg
Qwen 3.5-U	1.00	1.00	0.52	0.25	1.00	0.99	0.79
Qwen 3.5-MA	1.00	1.00	0.38	0.11	1.00	0.99	0.75
DeepSeek R1-U	0.99	0.99	0.99	0.81	0.89	0.83	**0.91**
DeepSeek R1-MA	1.00	1.00	0.15	0.03	1.00	0.99	0.70

Table 5: Mathematics domain results (n=150). U = Unified, MA = Multi-Agent.

4 Analysis & Discussion

Results reveal that neither model dominates universally across domains, and that architecture–task alignment is a stronger determinant of feedback quality than parameter count. Qwen 3.5 performs best on writing (avg $\approx$ 0.81), DeepSeek R1 excels on mathematics in unified mode (avg = 0.91), and programming challenges both models most severely (avg = 0.29–0.48).

Programming. The difficulty of programming feedback reflects a fundamental asymmetry in LLM capabilities: code generation and code understanding are distinct tasks (Haroon et al., 2025). While generation translates intent into code, diagnostic feedback requires reasoning about existing code semantics—a capability that benchmark scores do not capture. Semantic-preserving mutations cause LLMs to fail on previously localized faults in 78% of cases, revealing reliance on syntactic patterns over semantic understanding (Haroon et al., 2025), and systematic failures have been reported when LLMs verify code against specifications (Jin and Chen, 2025). Our findings provide direct educational evidence for this gap: fluency in explaining code does not imply understanding of code behavior. Notably, Qwen 3.5 outperforms DeepSeek on code generation benchmarks, yet DeepSeek achieves 66% higher feedback scores (0.48 vs. 0.29)—confirming that generation capability does not transfer to diagnostic tasks.

Writing. The results reveal a clear performance gap between Qwen 3.5 and DeepSeek (avg 0.81 vs. 0.35–0.38), with the gap most pronounced on accuracy: Qwen correctly identifies writing issues (WA = 0.93) and produces accurate suggestions (GA = 0.91), while DeepSeek scores only 0.16–0.18 on both. DeepSeek's high selectivity (WS $\approx$ 0.94) alongside near-zero accuracy reveals *avoidance behavior*—the model avoids criticism by missing issues entirely rather than correctly filtering them. This shows that high selectivity is not a reliable proxy for feedback quality. Prompting approach has minimal effect on either model, suggesting that writing assessment is a holistic task where model capability matters more than prompting architecture.

Mathematics. Both models achieve near-perfect error detection (MA, MS $\approx$ 1.0), confirming that identifying mathematical mistakes is tractable at this scale. The key differentiator is solution guidance: DeepSeek-Unified achieves SA = 0.81 vs. only 0.25 for Qwen, reflecting DeepSeek's reasoning advantage. However, the multi-agent approach severely degrades both models—DeepSeek drops from avg 0.91 to 0.70, with SA collapsing to 0.03. This challenges the assumption that task decomposition improves performance (Wang et al., 2025): mathematics requires a coherent reasoning chain, and splitting error diagnosis from feedback generation disrupts it.

Key takeaway. Multi-agent prompting is domain-conditional: neutral in programming, harmful in mathematics, and inconsequential in writing. It should not be adopted by default for educational feedback generation, and its benefit depends on whether the task is decomposable or requires a cohesive reasoning chain. Beyond prompting, domain characteristics strongly influence feedback quality. Mathematics proves most tractable (avg 0.70-0.91), followed by writing (0.35-0.81), with programming presenting the greatest challenge (0.27-0.48). This hierarchy suggests that structured domains with well-defined, step-by-step correctness criteria are easier to address learners' issues than tasks that require nuanced judgments. Model architecture is a stronger determinant of feedback quality than parameter count. DeepSeek R1's reasoning-optimized approach excels in structured domains that require logical analysis, while Qwen 3.5's general-purpose design works better for holistic evaluation tasks. This pattern cautions against a one-size-fits-all model selection across multiple domains. Finally, our results highlight that programming models can generate fluent feedback while failing to accurately identify bugs, confirming that code-generation capabilities do not transfer to code-diagnostic tasks.

5 Conclusion and Future Directions

We examine a single LLM-based model (as a unified evaluation framework) for feedback generation across three disciplines: (i) mathematics, (ii) academic writing, and (iii) programming. We evaluated two open-source models of different paradigms: reasoning-optimized (DeepSeek R1) and general-purpose (Qwen 3.5) under unified and multi-agent prompting techniques. Our findings reveal that open-source models show potential for cross-domain feedback generation, but with varying effectiveness. DeepSeek R1 excels in structured domains where step-by-step reasoning approaches are favored, such as mathematics (avg 0.91), while Qwen 3.5 performs better on holistic tasks such as writing (avg 0.81). Programming presents a challenge for both models (avg 0.27-0.48). Despite their remarkable code generation capabilities on standard coding benchmarks, they are not yet strong enough for complex code-diagnostic tasks. These results caution against using a single model-supported system across all assessment types, and highlight that model architecture is a stronger determinant of feedback quality than prompting strategy.

Our evaluation assesses feedback quality through human judgment; however, high-quality feedback does not necessarily result in greater learning gains. Future research should examine whether feedback generated by LLMs supports students in developing conceptual understanding and critical thinking skills, rather than directly providing solutions. Following recent work on AI tutoring systems (Murtazin et al., 2025) and students' tendency to rely on AI models for direct answers across writing, programming, and general learning contexts (Denny et al., 2024; Li and Ma, 2025; Ge et al., 2025), we plan to test and deploy our framework as a classroom tool and systematically measure its impact on student performance and learning outcomes.

Limitations

This study has several limitations. First, evaluation is based on offline human judgments rather than classroom deployment, so educational impact on student learning remains unverified, and it is unclear whether high-scoring feedback would help improve learning outcomes, enhance student understanding, and develop critical thinking skills. Second, the evaluation covers only two small open-source models (Qwen 3.5 27B and DeepSeek R1 32B) and three relatively small datasets: 200 programming samples (ConDefects), 55 writing samples (Exposia), and 150 mathematics samples (MathEDU) datasets, which limits the breadth of generalization. Our primary goal was to demonstrate a constrained environment to maximize the feasibility of replicating the proposed framework. Our findings may not generalize to larger open-source or proprietary models (e.g., GPT, Claude, Gemini). Although the datasets differ in their specific educational contexts and assessment formats, this diversity was intentionally used to assess cross-domain generalizability. Additionally, MathEDU was selected as the closest available dataset to reflect university-level mathematics, due to the absence of suitable calculus or high-level mathematics feedback dataset.

Third, our binary annotation scheme may not capture partial correctness or nuanced differences in feedback quality. Furthermore, our annotators were graduate students with domain expertise but no pedagogical training, which may have influenced their assessments of the pedagogical appropriateness of generated feedback. Inter-annotator agreement was calculated on a limited subset of overlapping samples, which may not fully capture annotation reliability across all domains and the full dataset. Finally, reference supervision differs across domains, especially in programming, where no gold feedback was available.

References

Areen Alnemrat, Hesham Aldamen, Mohamad Almashour, Mutasim Al-Deaibes, and Rami Al-Sharefeen. 2025. AI vs. teacher feedback on EFL argumentative writing: a quantitative study. *Frontiers in Education*, 10.

Esteban M. Aucejo and Kelvin Wong. 2025. The effect of feedback on student performance. *Journal of Public Economics*, 241:105274.

Anderson Pinheiro Cavalcanti, Arthur Barbosa, Ruan Carvalho, Fred Freitas, Yi-Shan Tsai, Dragan Gašević, and Rafael Ferreira Mello. 2021. Automatic feedback in online learning environments: A systematic literature review. *Computers and Education: Artificial Intelligence*, 2:100027.

Jorge Cisneros-González, Natalia Gordo-Herrera, Iván Barcia-Santos, and Javier Sánchez-Soriano. 2025. JorGPT: Instructor-aided grading of programming assignments with large language models (LLMs). *Future Internet*, 17(6):265.

Paul Denny, Stephen MacNeil, Jaromir Savelka, Leo Porter, and Andrew Luxton-Reilly. 2024. Desirable characteristics for ai teaching assistants in programming education. In *Proceedings of the 2024 on Innovation and Technology in Computer Science Education V. 1*, pages 408–414.

Bingyu Dong, Jie Bai, Tao Xu, and Yun Zhou. 2024. Large language models in education: A systematic review. In *2024 6th International Conference on Computer Science and Technologies in Education (CSTE)*, pages 131–134. IEEE.

Darren Paul Fisher, Gaelle Brotto, Iris Lim, and Colette Southam. 2025. The impact of timely formative feedback on university student motivation. *Assessment & Evaluation in Higher Education*, 50(4):622–631.

Wentao Ge, Yuqing Sun, Ziyan Wang, Haoyue Zheng, Weiyang He, Piaohong Wang, Qianyu Zhu, and Benyou Wang. 2025. Srlagent: Enhancing self-regulated learning skills through gamification and llm assistance. *arXiv preprint arXiv:2506.09968*.

Jovita Gilmore. 2025. Going beyond surface-level analysis: Using chatgpt to give feedback and praise for student writing. *English Australia Journal*, 41(1):85–90.

Sghaier Guizani, Tehseen Mazhar, Tariq Shahzad, Wasim Ahmad, Afsha Bibi, and Habib Hamam. 2025. A systematic literature review to implement large language model in higher education: issues and solutions. *Discover Education*, 4(1):35.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, and 1 others. 2025. Deepseek-r1 incentivizes reasoning in llms through reinforcement learning. *Nature*, 645:633–638.

Sabaat Haroon, Ahmad Faraz Khan, Ahmad Humayun, Waris Gill, Abdul Haddi Amjad, Ali R. Butt, Mohammad Taha Khan, and Muhammad Ali Gulzar. 2025. Assessing the impact of code changes on the fault localizability of large language models. *arXiv preprint arXiv:2504.04372*.

Muhammad Haseeb, Ahmad Imam Amjad, Maaz Amjad, and Victor S. Sheng. 2026. From code to rubrics: A multimodal evaluation of LLM systems for automated feedback generation. In *Companion Proceedings of the ACM on Web Conference (WWW)*. In press.

Helia Hashemi, Jason Eisner, Corby Rosset, Benjamin Van Durme, and Chris Kedzie. 2024. LLM-rubric: A multidimensional, calibrated approach to automated evaluation of natural language texts. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 13806–13834.

John Hattie and Helen Timperley. 2007. The power of feedback. *Review of educational research*, 77(1):81–112.

Wei-Ling Hsu, Yu-Chien Tang, and An-Zi Yen. 2026. MathEDU: Feedback generation on problem-solving processes for mathematical learning support. In *Proceedings of the 19th Conference of the European Chapter of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2883–2901, Rabat, Morocco. Association for Computational Linguistics.

Haolin Jin and Huaming Chen. 2025. Uncovering systematic failures of LLMs in verifying code against natural language specifications. In *Proceedings of the 40th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, page 3819–3823.

Annette Kinder, Fiona J. Briese, Marius Jacobs, Niclas Dern, Niels Glodny, Simon Jacobs, and Samuel Leßmann. 2025. Effects of adaptive feedback generated by a large language model: A case study in teacher education. *Computers and Education: Artificial Intelligence*, 8:100349.

Charles Koutcheme, Nicola Dainese, Sami Sarsa, Arto Hellas, Juho Leinonen, Syed Ashraf, and Paul Denny. 2025. Evaluating language models for generating and judging programming feedback. In *Proceedings of the 56th ACM Technical Symposium on Computer Science Education (SIGCSE)*, pages 624–630.

Yong Oh Lee, Byeonghun Bang, Joohyun Lee, and Sejun Oh. 2026. Personalized auto-grading and feedback system for constructive geometry tasks using large language models on an online math platform. *IEEE Access*, pages 17788–17802.

Huiyong Li and Boxuan Ma. 2025. Design of ai-powered tool for self-regulation support in programming education. *arXiv preprint arXiv:2504.03068*.

Michael Pin-Chuan Lin, Jing-Yuan Huang, Daniel H Chang, Gerald Tembrevilla, G Michael Bowen, Eric Poitras, Vasudevan Janarthanan, and Jeeho Ryoo. 2026. Open-source large language models in education: A narrative review of evidence, pedagogical roles, and learning outcomes. In *AI in Education*, page 4. MDPI.

Naiming Liu, Shashank Sonkar, Zichao Wang, Simon Woodhead, and Richard G. Baraniuk. 2023. Novice learner and expert tutor: Evaluating math reasoning abilities of large language models with misconceptions. *arXiv preprint arXiv:2310.02439*.

Martin Messer, Neil C. C. Brown, Michael Kolling, and Mia Shi. 2024. Automated grading and feedback tools for programming education: A systematic review. *ACM Transactions on Computing Education (TOCE)*, 24(1):1–43.

Kristina Murtazin, Oleg Shvets, Karl-Erik Karu, Viljam Puusep, Jelena Vendelin, Martijn Meeter, and Gunnar Piho. 2025. Student support during project-based learning using simulated automated formative feedback: an experimental evaluation. In *Frontiers in Education*, volume 10, page 1621644. Frontiers Media SA.

Fredrick Eneye Tania-Amanda Nkoyo, Chukwuebuka Fortunate Ijezue, Maaz Amjad, Ahmad Imam Amjad, Sabur Butt, and Gerardo Castañeda-Garza. 2025. 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)*, pages 477–498. Association for Computational Linguistics.

Nelly Rigaud Téllez, Patricia Rayón Villela, and Roberto Blanco Bautista. 2024. Evaluating chatgpt-generated linear algebra formative assessments. *International Journal of Interactive Multimedia and Artificial Intelligence*, 8(5):75–82.

Yaoxiang Wang, Zhiyong Wu, Junfeng Yao, and Jinsong Su. 2025. Tdag: A multi-agent framework based on dynamic task decomposition and agent generation. *Neural Networks*, 185:107200.

Yonghao Wu, Zheng Li, Jie M. Zhang, and Yong Liu. 2023. Condefects: A new dataset to address the data leakage concern for LLM-based fault localization and program repair. *arXiv preprint arXiv:2310.16253*.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, and Others. 2025. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.

Boyang Yang, Haoye Tian, Weiguo Pian, Haoran Yu, Haitao Wang, Jacques Klein, Tegawendé F. Bissyandé, and Shunfu Jin. 2024. Cref: An llm-based conversational software repair framework for programming tutors. In *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA)*, pages 882–894.

Dennis Zyska, Alla Rozovskaya, Ilia Kuznetsov, and Iryna Gurevych. 2026. Exposía: Teaching and assessment of academic writing skills for research project proposals and peer feedback. *arXiv preprint arXiv:2601.06536*.

Figure A.1. Unified Prompt Template

You are an experienced computer science instructor reviewing a student's buggy code submission. Your goal is to provide feedback that helps the student understand and fix the bug on their own.

Below is a problem description and an incorrect program written by a student.
⟨problem description⟩
⟨language⟩
⟨student code⟩

Before writing your feedback, reason through the following steps internally:

1. Identify the exact location and type of the primary bug.

2. Determine the conceptual misunderstanding that likely caused it.

3. Decide how to guide the student toward the fix without revealing the solution.

Then write your feedback following these rules:

- Start with one sentence acknowledging something the student did correctly.

- Clearly identify the bug location (e.g., line number) and its error type.

- Explain the underlying conceptual misunderstanding in plain language.

- Describe the fix direction (e.g., "change X to Y") without providing corrected code.

- Do not flag issues that do not exist in the student's code.

Output: 150–250 words, 3–4 short paragraphs, no markdown headers or code blocks.

Figure 2: Unified prompt for programming feedback generation.

Figure A.2. Writing Unified Prompt Template

You are an academic writing expert evaluating a Bachelor's thesis research proposal. Your role combines subject-matter review and writing instruction.

Below is a Bachelor's thesis research proposal written by a student.

⟨expose text⟩

Evaluate the proposal across four dimensions using the rubric below. For each criterion:

1. Locate the relevant passage in the text.

2. Apply the rubric to what is explicitly written — do not assume missing content.

3. Assign an integer score within the allowed range.

4. Ensure scores match your written analysis.

Dimensions and maximum scores:

- Problem & RQ (MO1–MO7): max 9 pts

- Literature & Theory (AP1–AP7, BI1–BI2): max 13 pts

- Methodology (AP8, MD1–MD7): max 8 pts

- Writing Quality (LA1–LA2): max 4 pts

Scoring rules: assign only integers within the defined range; score based on development, not mere mention; use only evidence explicitly present in the text.

Output format:

SCORES: MO1:[s] | MO2:[s] | ... | MO7:[s]
AP1:[s] | ... | AP7:[s] | BI1:[s] | BI2:[s]
AP8:[s] | MD1:[s] | ... | MD7:[s] | LA1:[s] | LA2:[s]

FEEDBACK: [150–250 words. Strengths first, then key areas for improvement.]

Figure 3: Unified prompt for academic writing feedback and scoring.

Figure A.3. Mathematics Unified Prompt Template

You are an experienced mathematics tutor reviewing a student's incorrect solution. Your goal is to help the student identify and understand their mistake so they can correct it independently.

Below is a mathematics problem and a student's incorrect solution.
⟨problem⟩
⟨student answer⟩
⟨student process⟩

Before writing your feedback, reason through the following steps internally:

1. Identify the step in the student's process where the error first occurs.

2. Determine whether the error is conceptual, procedural, or computational.

3. Decide how to guide the student without simply giving the answer away.

Then write your feedback following these rules:

- Acknowledge any correct steps the student took before the error, if applicable.

- Clearly identify where and what the mistake is, using the student's own work as reference.

- Explain the mathematical misunderstanding in plain language at the student's level.

- Guide the student step by step and state the correct answer at the end.

- Do not penalize valid alternative solution paths.

- Do not assume reasoning that is not shown in the student's process.

Output: FEEDBACK: [50–120 words. Correct work (if any) → mistake → guidance → correct answer.]

Figure 4: Unified prompt for mathematics feedback generation.

Figure A.4. Multi-Agent Prompt Template (Agent – A)

You are a precise code diagnostician. Your only job is to identify the primary bug in a student's code submission.

Analyze the following student code against the problem statement.
⟨problem description⟩
⟨language⟩
⟨student code⟩

Your tasks are as follows:

1. Identify the single most critical bug that explains why the code fails. If multiple bugs exist, focus on the root cause.

2. Do not provide student-friendly feedback — only a structured technical diagnosis.

3. Output ONLY the following format with no extra commentary:

BUG LOCATION: ⟨line number or segment, e.g. "Line 5: n = n / 2"⟩

ERROR TYPE: ⟨Logic Error | Off-by-One | Incorrect Operator | Wrong Variable | Missing Condition | Type Error | Other⟩

ROOT CAUSE: ⟨one sentence: what the code does vs. what it should do⟩

STUDENT MISCONCEPTION: ⟨one sentence: the likely conceptual gap⟩

FIX DIRECTION: ⟨one sentence: the correction, without writing corrected code⟩

Figure 5: Error Analyzer prompt for the multi-agent programming pipeline.

Figure A.5. Multi-Agent Prompt Template (Agent – B)

You are an experienced computer science instructor. You have received a structured bug analysis for a student's submission. Your task is to convert this technical diagnosis into clear, encouraging, and pedagogically useful feedback.

Below is a bug analysis for a student's code submission.
⟨problem description⟩
⟨language⟩
⟨student code⟩
⟨bug analysis⟩

Your tasks are as follows:

1. Open with one sentence acknowledging something the student did correctly.

2. Pinpoint the bug location and name its error type in plain language.

3. Explain the conceptual misunderstanding behind the bug clearly.

4. Guide the student toward the fix (e.g., "change X to Y") without providing corrected code.

Please follow these rules:

- Ground every claim strictly in the bug analysis — do not invent errors not identified.

- Use accessible language suitable for an introductory programming student.

- Do not write code blocks or corrected code in your output.

Output: 150–250 words, 3–4 short paragraphs, no markdown headers or code blocks.

Figure 6: Feedback Generator prompt for the multi-agent programming pipeline.