Aligning Small Language Models for Programming Feedback

Towards Scalable Coding Support in a Massive Global Course

Charles Koutcheme
charles.koutcheme@aalto.fi
Aalto University
Espoo, Finland

Juliette Woodrow
jwoordrow@stanford.edu
Stanford University
Palo Alto, United States

Chris Piech
cpiech@stanford.edu
Stanford University
Palo Aalto, USA

Abstract

Providing timely and actionable feedback is essential for students learning to program. While large language models (LLMs) are increasingly used to automate this process, they remain costly to deploy and raise concerns around privacy and institutional control. Small language models (SLMs) offer a promising alternative: they can be run locally and integrated more flexibly into educational platforms. However, their out-of-the-box performance is often poor, requiring targeted training to be effective in classrooms. In this paper, we investigate whether a trained 3B-parameter SLM, guided by rubric-based prompting and a pipeline combining supervised and preference-based learning, can generate diagnostic feedback that approaches the quality of larger models. We deploy the model in a large-scale online programming course and compare its feedback to its base and fine-tuned variants, Llama-3.1-8B, and GPT-4.1, using human ratings from 53 teaching assistants and an automated LLM-as-a-judge analysis. Our results show that careful training narrows the feedback quality gap between an SLM and an LLM from over 80 to just 10 percentage points on key metrics. The trained SLM more rarely hallucinates errors, is often rated as helpful by educators, and only occasionally misses issues in student code. These findings suggest that small models can serve as practical and scalable targeted feedback solutions in large educational settings, while LLMs may remain necessary for more comprehensive diagnostic feedback.

CCS Concepts

• **Social and professional topics → Computing education**.

Keywords

programming feedback, large language models, small language models, generative AI, open source, automatic feedback

ACM Reference Format:
Charles Koutcheme, Juliette Woodrow, and Chris Piech. 2026. Aligning Small Language Models for Programming Feedback: Towards Scalable Coding Support in a Massive Global Course. In *To appear in Proceedings of the 57th ACM Technical Symposium on Computer Science Education (SIGCSE TS 2026), February 18–21, 2026, St. Louis, Missouri, USA.* ACM, New York, NY, USA, 7 pages.

SIGCSE TS 2026, February 18–21, 2026, St. Louis, Missouri, USA
2026.

1 Introduction

Learning to program is challenging for many. These challenges can be somewhat alleviated with improved teaching practice [30]. A key part of this is providing feedback, which should be timely and accurate [28]. Large language models (LLMs) have shown exceptional success in providing such feedback [22], leading to their growing adoption in classrooms [1, 10, 20, 29, 31]. However, relying on third-party services that provide access to LLMs can introduce cost obstacles, privacy concerns, and scalability issues [5, 6].

These constraints are driving a growing shift towards using smaller, open-source models [12], which can be deployed locally [16, 34] to reduce costs and give educators greater control over their students' data. Such models are particularly promising for providing timely feedback in large-scale classes such as Massive Open Online Classes (MOOCs), which feature thousands of students working on hundreds of exercises.

Although small language models (SLMs) show promise for scalable and private deployment, recent studies demonstrate that their feedback quality often falls significantly short of LLMs [17, 18]. SLMs are far more likely to produce inaccurate or hallucinated feedback, which can mislead learners and limit the pedagogical value of automated feedback systems. Improving their performance is therefore essential before they can be reliably deployed in real-world educational settings.

This study contributes to understanding the tradeoffs educators face when replacing a large language model with a smaller, more deployable model. While prior work has benchmarked trained small language models on controlled datasets for generating hints [16], it remains unclear how they perform in large-scale settings for other types of feedback. Specifically, we assess their ability to generate *diagnostic feedback*, that is, feedback that helps students understand both what went well and what went wrong in their final programs. We address the following research question (**RQ**):

> **What is the performance tradeoff in diagnostic feedback quality when replacing a large language model with a trained small language model?**

To answer this question, we adopt a training framework that transfers feedback capabilities from a large language model (GPT-4.1) to a small language model (Qwen-2.5-Coder-3B, 3B parameters). We deploy the trained model within *Code In Place*, a large-scale online Python programming course, to generate feedback on four programming exercises completed in the final week of the course. We compare its feedback to that of larger models, including GPT-4.1 and Llama-3.1-8B, using two complementary evaluation methods: (1) expert assessments by teaching assistants, who rate feedback for correctness and helpfulness, and (2) automated judgment using an LLM-as-a-judge framework.

Our results show that, while untrained small language models lagged far behind LLMs, by more than 80 percentage points on key feedback metrics, targeted training can close most of this gap, reducing the difference to just 10 percentage points compared to GPT-4.1-level feedback quality. In particular, our trained SLM rarely hallucinated errors and was frequently rated as helpful by educators, although it sometimes missed issues in student programs. These findings mark a meaningful step toward scalable, high-quality programming education, where small, locally deployable models could eventually support learners directly on their own devices.

Our study also makes the following technical contributions:

- We introduce a rubric-based prompting method to align language models' feedback with instructional goals.
- We design and evaluate a training pipeline that combines supervised fine-tuning with preference learning, and analyze its impact on the feedback quality of small language models in a large-scale, open-access programming course.
- We release our code for training small language models for diagnostic feedback: cip25-aiep

2 Related Work

While some prior work has fine-tuned large, proprietary models for feedback-related tasks [21], our study builds upon recent efforts to fine-tune open-source language models using preference-based methods such as Direct Preference Optimization (DPO) [26], a technique that trains models through ranking or pairwise comparisons to encourage preferred outputs. In programming education, Kumar et al. [2] use DPO to train models for Socratic debugging and dialogue, while Hicke et al. [12] combine supervised and preference-based approaches to support retrieval-augmented generation for answering student forum questions. Similar preference-based methods have also been used in other educational domains: Woodrow et al. [32] and Scarlatos et al. [27] apply supervised finetuning and DPO to provide feedback in probability and mathematics courses, leveraging preference data collected from humans and LLMs. These studies collectively demonstrate the broad applicability of reinforcement learning and preference-based fine-tuning for educational feedback, though none address diagnostic programming feedback. Closer to our work is Kotalwar et al. [16], who fine-tune small language models to generate Socratic hints using LLM-generated data and supervised learning. However, our work differs in three ways: (1) they do not investigate diagnostic feedback, (2) their approach does not incorporate preference learning and, (3) their evaluation is not conducted in a large-scale course setting.

3 Context

We conduct our study in the context of *Code In Place*[1], a large-scale MOOC designed to teach students worldwide the fundamentals of Python programming. The course is primarily intended for people who have no prior programming experience. The course runs annually over six weeks and combines pre-recorded lectures, hands-on coding exercises, and weekly live Zoom sessions. Each session is led by a volunteer TA, with one TA assigned to every ten students, creating a supportive peer-driven learning community.

[1]https://codeinplace.stanford.edu/

By the end of the course, students are expected to grasp core programming concepts such as control flow, loops, console input/output, graphics, lists, and dictionaries. Our study takes place during the fifth edition of the course, offered in spring 2025.

In the final week, students are invited to complete four voluntary programming assignments, referred to as *diagnostic exercises*, designed to assess their mastery of the course content. The exercises include: (1) determining astronaut height eligibility, (2) identifying even or odd numbers from 1 to 100, (3) processing a non-decreasing number sequence, and (4) fixing a buggy graphics program to draw two cars at given positions. These exercises are intended to consolidate key skills learned throughout the course, including variables, loops, input/output, and graphics. To mimic an exam setting, students are given three hours to complete the exercises, during which they can run their code but have no access to unit tests or automated grading. We consider only their last submitted program.

Our goal is to provide students with *diagnostic feedback*, which clearly highlights both strengths and errors in a student's submission in a constructive tone, going beyond basic test-case feedback. Such feedback can prevent discouragement in struggling students while reinforcing confidence in those who succeed [9].

Historically, feedback on diagnostic exercises in early editions of our MOOC was powered by teaching assistants completing grading rubrics [33]. Each rubric consisted of items aligned with key aspects of the intended problem-solving strategy, scored on a Likert-like scale and accompanied by representative errors. Teaching assistants were asked to justify their selections with short, one-line explanations. Figure 1 shows an example rubric.

Example Grading Rubric

- **Looping:** Correct / Minor error (e.g., off-by-one) / Major issue
- **Checking Even or Odd:** Correct / Minor error / Major issue
- **Printing:** Correct / Minor formatting / Major issue
- **Syntax Errors:** None / Minor / Major

Figure 1: Illustrative version of the grading rubric used to assess the even or odd exercise.

Given how tedious this task can be for TAs, we began exploring the use of generative language models to support the teaching team. Prior to the advent of powerful LLMs, a popular line of work framed programming feedback as a multi-label classification task, with each label corresponding to a mistake or success pattern defined in grading rubrics [23, 33]. Building on this foundation, we adapt a rubric-based structure to prompt modern language models for programming feedback generation. We found rubrics benefit both learners and educators: (1) they provide a structured scaffold that aligns feedback with each exercise's learning objectives, potentially mitigating hallucinations [11], and (2) they allow instructors to quickly identify patterns of student difficulties at scale. In the 2023 (resp. 2024) edition of the course, we piloted the use of GPT-3.5-turbo (resp. GPT-4o) to automatically fill in rubrics and generate short explanations, using one-shot examples based on human-annotated feedback. Building on this experience, we now investigate whether trained small language models can also support the provision of rubric-based diagnostic feedback and offer a more cost-effective and scalable solution for future course offerings.

4 Methodology

In this section, we revisit the task of diagnostic feedback, and then present our training methodology for improving SLMs.

4.1 Generating Feedback

To elicit high-quality diagnostic feedback, we refined the rubric-based prompting approach used in prior course deployments, incorporating observations from earlier pilot studies and recent advances in language model prompting. Figure 2 illustrates our feedback prompt template.

We provide a language model with the problem description, the student code, and the grading rubric as input and instruct the language model to provide diagnostic feedback using a structured approach described below.

Step 1 - Reasoning. First, the language model is asked to reason about the student's mistakes and successes using the provided rubric. Specifically, we adopt zero-shot chain-of-thought [15] (i.e., we pre-fill the model answer with *"Let's think step-by-step"*) to enforce the model to lay out its analysis while allowing space to reflect freely without phrasing sentences that would need to be understandable by students.

Step 2 - Grading. Next, the language model is instructed to fill in the grading rubric by selecting the appropriate option for each item. This step consolidates the model's reasoning into a structured decision, scaffolding the feedback generation process. Moreover, it benefits educators by enabling systematic comparison of student submissions and analysis of common errors at scale.

Step 3 - Feedback. Finally, the model provides feedback addressing each student directly. We explicitly instruct the model to (1) begin its feedback by highlighting all positive items in the student submission, (2) explain the first two mistakes in their code according to the grading rubric, and (3) finish with encouragements. This strategy, sandwiching negative feedback between positive praises, is meant to ensure feedback is received constructively [3]. During our pilot studies, we observed that students who failed multiple rubric items often received lengthy negative feedback. Such feedback could become overwhelming and demotivating. To address this, our prompt explicitly instructs the language model to focus negative feedback on only the first two mistakes, following the order of items in the rubric, which reflects the logical progression of the intended problem-solving strategy. While this approach does not address all mistakes in a student's submission, we believe it is preferable for students to focus on a few key issues rather than risk disengagement due to information overload [8, 28].

4.2 Fine-tuning Small Language Models: Distilling Knowledge From LLMs

In this subsection, we formalise our approach to fine-tune a small language model π_θ (the *student* LM) to provide high-quality diagnostic feedback. Let us assume access to a dataset $\mathcal{D} = \{(d^i, s^i, r^i)\}_{i=1}^N$ consisting of N triplets of problem descriptions d^i, student programs s^i, and rubrics r^i. We also assume access to a *teacher* LLM, π_τ, available via an online API (e.g., GPT-4o via the OpenAI API), which is proficient at providing feedback but expensive to query.

In this work, we propose to transfer some of the teacher LLM's feedback capabilities into the small model using a combination of supervised fine-tuning and preference-based learning.

Feedback prompt template

Inputs: Problem description, student code, grading rubric

Outputs:

(1) **Reasoning:** Reflect on the quality of the student's submission, considering each rubric item in order.
(2) **Rubric grading:** For each rubric item, choose the rubric option that corresponds to it.
(3) **Feedback:**
- Highlight all satisfied rubric items.
- Identify and explain only the first two mistakes.
- Ensure feedback is clear, encouraging, and actionable.

Figure 2: We prompt our language models to provide feedback using rubric scaffolding and a chain of thought approach.

Step 1- Supervised fine-tuning. Given the lack of human annotations, following Kotalwar et al.[16], we leverage supervised fine-tuning (SFT) using feedback generated by a teacher model. For each incorrect program in the training set, we use the structured prompt described in Section 4.1 to obtain teacher outputs $(t_\tau^i, g_\tau^i, f_\tau^i) \sim \pi_\tau(s^i, d^i, r^i)$, where t_τ^i is the model's reasoning (thought), g_τ^i the filled-in rubric, and f_τ^i the generated feedback. We generate such feedback using greedy decoding. The student model is then fine-tuned using a standard negative log-likelihood (NLL) loss over these outputs, yielding the SFT model π_{sft}. SFT acts as a basic form of knowledge distillation, where the student imitates the teacher.

Step 2 - Preference-based fine-tuning. To further improve the model, we use a variant of Direct Preference Optimization (DPO) [26], an offline algorithm that updates models based on ranked output pairs. In our setting, this requires constructing a preference dataset for training: $\mathcal{P} = \{(x^i, y_{\text{win}}^i, y_{\text{lose}}^i)\}_{i=1}^M$, where $x^i = (d^i, s^i, r^i)$ is the input, and each output $y^i = (t^i, g^i, f^i)$ consists of a thought process, grading decision, as well as written feedback, with y_{win}^i being preferred over y_{lose}^i in terms of overall feedback quality. Prior work built such datasets by generating multiple outputs per input and ranking them via human annotators [32] or LLM-as-a-judge systems [27]. While very effective, these methods can be costly to scale.

Instead, we design our approach to squeeze as much knowledge as possible from the available teacher generations. To that end, we automate preference construction by treating the teacher's output as the preferred response y_{win}^i and the student SFT output as the dispreferred one y_{lose}^i. This strategy aligns with recent work in programming education that leverages existing human annotations to create preference pairs [2, 12]. We generate the SFT output $(t_{\text{sft}}^i, g_{\text{sft}}^i, f_{\text{sft}}^i) = \pi_{\text{sft}}(x^i)$ using greedy decoding.

Prior work suggests that language models trained with the standard DPO loss [26] in settings where both outputs are syntactically similar can sometimes struggle to capture meaningful learning signals, leading in turn to degraded performance [4], a trend we also observed in our preliminary experiments using teacher LLM and SFT model outputs.

To address this, we adopt the Noise Contrastive Alignment (NCA) loss [4], a variant of DPO that provides a more stable learning signal. NCA has shown improved results on math and coding tasks by enforcing alignment with the preferred output while still accounting for the dispreferred one. We refer readers to the original work for technical details.

4.3 Experimental Setup

We use GPT-4.1 as the teacher LLM (π_τ), selected based on OpenAI's benchmarks showing stronger performance on programming tasks compared to GPT-4o. For the student model (π_θ), we fine-tune Qwen-2.5-Coder-3B [14], a 3 billion parameter language model that currently leads performance among *non-reasoning* models in its size class on several open-source benchmarks. We deliberately chose a *student* model of this scale to reflect our long-term goal of deploying feedback models directly on students' devices. Models in the 3B range offer a practical tradeoff between accuracy, inference speed, and memory footprint.

Our training dataset ($\mathcal{D}$) consisted of all collected student submissions for the *fourth edition* of our MOOC, with roughly 1,000 unique submissions per exercise (after duplicate elimination with AST normalization). We fine-tune the student model using LoRA [13], a parameter-efficient finetuning (PEFT) technique that trains only a small set of additional parameters, keeping the base model frozen. We report our training hyperparameters in our released code base.

5 Evaluation

After training, we collected student submissions from the diagnostic exercises and generated feedback for each solution using both the trained SLM and GPT-4.1. Feedback generation followed the approach described in Section 4.1. We focus on two complementary evaluation perspectives presented in this section: (1) expert evaluations from teaching assistants and (2) automated assessment using an LLM-as-a-judge, both aimed at answering our (**RQ**).

5.1 Feedback Quality Criteria

Butler et al.[3] suggest that effective feedback involves two main stages: first, *noticing* mistakes, and second, *responding* to those errors. Drawing on this perspective and recent work in educational feedback [1, 18, 27, 32], we adopt *correctness* and *helpfulness* as the primary criteria guiding our evaluation. Specifically, correctness reflects whether the feedback accurately identifies *the first two errors* in the student program (while highlighting the strengths of the solution); helpfulness captures whether the information is communicated in a way that meaningfully supports student understanding and learning. By definition, incorrect feedback cannot be helpful [25]; however, even accurate feedback may be unhelpful if, for instance, it uses unfamiliar concepts or is not beginner-friendly [11].

To enable a more detailed analysis, we further decompose correctness into four dimensions drawn from prior work [19, 27]: *perceptivity* (whether at least one mistake is identified), *accuracy* (whether the first two mistakes are correctly identified), *selectivity* (whether the feedback avoids hallucinating errors not present in the student's program), and *positivity* (whether the feedback maintains a constructive tone and highlights correct aspects of the solution).

We do not decompose the helpfulness criterion, and instead, we rely on human annotators to assess it; We use high-level correctness and helpfulness as the two main criteria in our human evaluation and leverage the detailed criteria in our LLM evaluation.

5.2 Human Evaluation

We recruited volunteer teaching assistants to evaluate the quality of the feedback. Each TA was assigned five student submissions per exercise, sampled using Malik et al. [23] zipf-aware selection strategy to ensure coverage of both typical and atypical solutions. For each student submission, TAs were shown two anonymized feedback responses, one from each model, in a randomized order to ensure blindness. They rated each response using our two main quality criteria: *correctness* and *helpfulness* (both binary).

Following Woodrow et al. [32], they also indicated which feedback they would prefer to provide to a student, or whether they liked both (or neither) responses. This dual approach captures both objective quality judgments and subjective pedagogical preferences. TAs also had the option to add comments to justify their annotation.

5.3 LLM Evaluation

To scale our evaluation, we adopt an LLM-as-judge approach [35], which has proven reliable in prior work for assessing feedback quality at scale [12, 18, 19, 27, 32]. We use this setup to contextualize the performance of our trained model (NCA), comparing it against two baselines: its base version (Qwen-2.5-Coder-3B) and the supervised fine-tuned version (SFT). We also include Llama-3.1-8B [7] as a reference point, given its widespread use in recent studies training language models for educational feedback [2, 16, 27, 32].

Our evaluation follows the Ground-truth Assisted Grading (GAG) strategy from Koutcheme et al. [18], in which an LLM (here, GPT-4.1) compares a model-generated feedback response to a trusted reference. Since we do not have human-written references, we use GPT-4.1-generated feedback that has been independently validated by teaching assistants as both correct and helpful, ensuring the reference feedback is of high quality. As a reminder, GPT-4.1-generated feedback is an upper bound on the quality of feedback we can obtain because the SLM is trained from outputs of the LLM. While this setup limits our evaluation to submissions where GPT-4.1 produces high-quality responses, it offers a more reliable way to answer our research question and understand the performance tradeoff when replacing GPT-4.1 with a trained small language model. Specifically, given a problem description, student code, and this validated reference feedback, we prompt GPT-4.1 to judge the evaluated feedback using the detailed correctness quality criteria outlined earlier: *perceptivity*, *accuracy*, *selectivity*, and *positivity*.

6 Results

In total, 5,452 students completed the diagnostic exercises. We had 53 TAs volunteering to annotate a total of 1060 student submissions, resulting in 265 annotations for each exercise.

6.1 Human Evaluation: Absolute Quality

Figure 3 shows the performance of our trained model (Qwen-2.5-3B NCA trained) against GPT-4.1, as evaluated by our teaching assistants across our exercises. We can make the following observations:

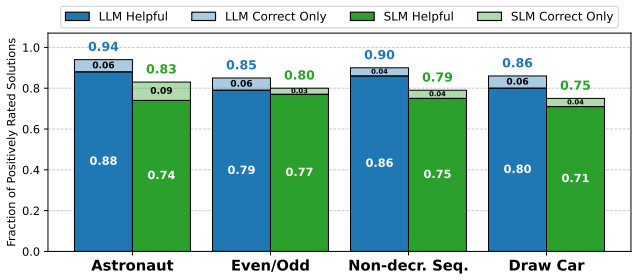

Figure 3: Proportion of positively rated feedback from the LLM (GPT-4.1) and the SLM (Qwen-2.5-Coder-3B, NCA-trained) across four diagnostic exercises, as evaluated by teaching assistants for correctness and helpfulness.

In terms of correctness, the gap between our trained SLM and GPT-4.1 remains relatively consistent: GPT-4.1 produces, on average, 11 percentage points more correct and helpful feedback. An exception is the even/odd exercise, where the gap narrows to just 4 percentage points. Both models perform best on the draw-car exercise. GPT-4.1 struggles most with even/odd, while our SLM performs worst on the non-decreasing sequence task.

We observe similar trends when analyzing helpfulness. As a reminder, feedback can be correct without being helpful, but helpful feedback must be correct. GPT-4.1 shows a small drop in helpfulness compared to correctness: losing 0.06 percentage points on the draw-car, even/odd, and non-decreasing sequence exercises, and 0.04 on the astronaut exercise. The small language model exhibits an average drop of 0.04 points on even/odd, non-decreasing sequence, and draw-car. However, it generates significantly less (0.09 percentage points lost) helpful feedback than correct feedback on the astronaut exercise.

6.2 Human Evaluation: Subjective Preferences

Figure 4 presents teaching assistants' preferences between LLM and SLM feedback, considering only cases where both responses were rated correct and helpful.

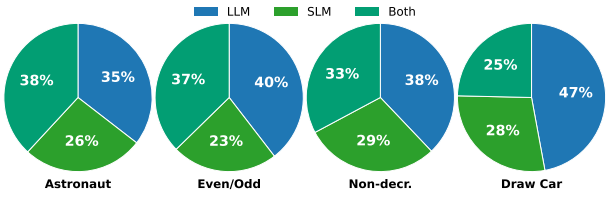

Figure 4: Teaching assistants' preferences between LLM and SLM feedback for responses rated as both correct and helpful.

Our results suggest that for the Astronaut, Even/Odd, and Non-Decreasing Sequence exercises, SLM feedback is *at least* as preferred as LLM feedback in roughly 65% of cases. The only exception is the Draw Car exercise, where only 55% of feedback from the SLM meets this threshold, indicating a more noticeable tradeoff when replacing the LLM with the SLM.

To better understand TA preferences, we conducted a comparative linguistic analysis of their comments. Using GPT-4.1, we annotated each comment with a normalized superlative adjective (e.g., "more clear", "more generalizable"), then standardized them into a concise set (e.g., "clear", "concise", "supportive"). We then computed the standardized comments frequencies across responses preferring LLM vs. SLM feedback and calculated log odds ratios (with Laplace smoothing) [24] to identify qualities distinctive of each model.

In order of importance, TAs described LLM feedback as: more *complete*, *friendly*, *supportive*, *positive*, and *specific*. In contrast, SLM feedback was more often characterized as more *generalizable*, *less harsh*, more *constructive*, and more *detailed*.

These differences suggest that, even after LLM distillation, the SLM retains a distinctive feedback writing style. Zhou et al. [36] posits that fine-tuning *reorganizes existing knowledge* to match task expectations, but that most of a model's core capabilities come from pretraining. This might explain why some TAs prefer one feedback over the other even when both are perfectly rated.

6.3 LLM Evaluation

After evaluating the model's overall performance with expert judgment, we now turn to a more detailed analysis of *correctness* using an LLM-as-a-judge framework. Figure 5 presents our results.

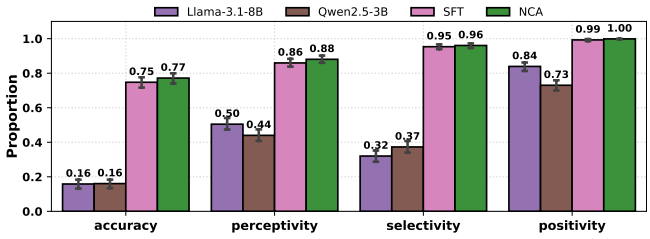

Figure 5: Proportion of feedback generated by four small models that satisfy each criteria as judged by GPT-4.1.

Smaller language models (Qwen-2.5-3B and Llama-3.1-8B) struggle to generate high-quality feedback. Accuracy remains below 20%, indicating that these models rarely identify both of the first two errors in student code. Although perceptivity is slightly higher (around 44–50%), it still falls short of a reliable baseline. Combined with low selectivity scores (below 40%), these findings confirm that untrained SLMs do indeed hallucinate issues [11, 18]. Interestingly, Qwen-2.5-3B performs comparably to Llama-3.1-8B in accuracy, reflecting the advances of recent 3B-scale models. Additionally, Qwen is generally more selective than Llama. However, its lower perceptivity suggests the 3B parameter model adopts a more conservative approach to error identification.

Training yields substantial performance gains across all criteria. Supervised fine-tuning (SFT) alone raises accuracy to 75% and selectivity to 95%, while preference learning with NCA provides additional improvements, pushing accuracy to 77% and selectivity to 96%. These incremental gains suggest that even when reusing the same teacher-generated data, preference-based optimization can help refine model behavior. However, broader improvements may require exposing the model to a more diverse range of generations or learning scenarios [27].

The trained SLM achieves 88% perceptivity, typically identifying at least one meaningful issue in most submissions, and over 94% selectivity, rarely hallucinating errors. This suggests that the trained SLM could be particularly effective at generating hints or targeted feedback on single issues. Our 3B model achieves comparable correctness to Kotalwar et al. [16] 8B model trained for hint generation, reflecting the steady progress of small open-source models.

Looking closer, the lower overall accuracy and very high positivity suggest a cautiousness in flagging errors, with the model sometimes missing mistakes. Put differently, if we view feedback generation as multiple binary classification tasks [33], the model tends to make more false negatives than false positives which, in our MOOC setting, is the least harmful type of error as it avoids discouraging first time programming students. We observed earlier that our SLM exhibited already higher selectivity and lower perceptivity than LLama before training. Those observations reinforce our hypothesis that certain base model properties, such as caution in error identification or generation style (e.g. supportiveness) may remain even after supervised fine-tuning and preference learning.

6.4 Case Study: Student Perceptions

We deployed feedback to all 5,452 students who completed the diagnostic exercises. Each student was randomly assigned feedback from either the LLM or the SLM for three of the four assignments (Astronaut, Even/Odd, and Non-decreasing), while all students received LLM feedback for the Draw-Car exercise due to an implementation error. When viewing their feedback, students could optionally indicate whether it helped them *learn something* by giving a thumbs up or thumbs down. Students were not required to view or respond to their feedback. In total, 313 students responded. Almost all of them gave positive ratings across all exercises and models (at least 95% positive in each), indicating that students generally perceived the feedback as helpful regardless of model source.

Interpreting these results requires caution: participation was optional, the missing data is not random, and students may prefer praise over constructive criticism. Despite these limitations, the student ratings broadly mirrored TA preferences, with a slight preference for LLM feedback over SLM feedback.

7 Discussion

Implications for teaching and learning. Our findings suggest that modest training significantly boosts small language model (SLM) performance. While SLMs are not yet full replacements for LLMs in generating comprehensive diagnostic feedback, they can still play a valuable role in large-scale learning environments when deployed alongside LLMs. For example, SLMs can support lightweight tasks such as providing timely Socratic hints or explaining a single error during the coding process. These models can be deployed locally using tools like WebLLM, enabling fast, low-latency feedback, even offline [16]. In contrast, LLMs can be reserved for more detailed feedback when students complete their exercises. Trained SLMs can also be used for full diagnostic feedback, particularly on exercises where they perform well. Identifying such exercises could rely on LLM-as-judge evaluation [35], or emerging methods that relate a model's program repair capabilities to its feedback generation quality [17].

Importantly, our results also highlight the importance of base model selection. Educators should consider not only initial model performance but also the generation style of the model, as some characteristics can remain even post-training.

Limitations. Our study has several limitations. First, our evaluation focuses on a single small language model and a single large language model, within the context of one Python MOOC and just four exercises. While our findings are consistent with prior work, their broader generalization should be confirmed through replication studies in other courses and contexts. Future studies should also assess the effectiveness of newer small and large *reasoning models* for feedback generation. We also did not directly compare our rubric-driven prompting approach to a more open-ended baseline. In this work, rubric-based prompting is treated as an integral part of the feedback generation strategy, rather than a variable under evaluation. Future work should isolate and investigate the specific contribution of rubrics prompting. Moreover, our prompts did not incorporate program execution results or unit test information [25], instead relying solely on model-based correctness detection. Including unit test outcomes could help filter out correct solutions or provide more targeted guidance to models about which aspects of student code need improvement. Finally, we did not assess inter-rater reliability among the course staff which may affect the consistency of subjective judgments.

Future Work. For future work, we plan to explore advanced training strategies to further narrow the performance gap between SLMs and LLMs. One direction involves applying human preference optimization techniques [32], using the large volume of TA preference data collected in this study to refine the model for future course iterations. In parallel, we aim to leverage the broader dataset spanning all exercises in our MOOC to support generalization and to explore additional points in the course where feedback models could be effectively deployed. Ultimately, our goal is to deploy these models directly in students' development environments, evaluate their impact on learning outcomes, and compare these results to studies using LLM-based feedback [6].

8 Conclusion

This work aimed to answer the following research question: ***What is the performance tradeoff in diagnostic feedback quality when replacing a large language model with a trained small language model?*** Our results suggest that while replacing an LLM with a trained SLM may lead to some issues in student solutions being omitted, the feedback provided by SLMs is still often perceived as helpful by educators, as it typically highlights at least one key issue in the student's program and more rarely hallucinates errors. These findings highlight the potential of hybrid systems that combine the efficiency of small models with the advanced capabilities of LLMs, supporting scalable feedback in large educational settings.

Acknowledgments

We thank all the teaching assistants who participated as annotators in our study. The full list of contributors is available in our project repository: cip25-aiep. This work was in part supported by Research Council of Finland grant #367787

References

[1] Umair Z. Ahmed, Shubham Sahai, Ben Leong, and Amey Karkare. 2025. Feasibility Study of Augmenting Teaching Assistants with AI for CS1 Programming Feedback. In *Proceedings of the 56th ACM Technical Symposium on Computer Science Education V. 1* (Pittsburgh, PA, USA) *(SIGCSETS 2025)*. Association for Computing Machinery, New York, NY, USA, 11–17. https://doi.org/10.1145/3641554.3701972

[2] Nischal Ashok K. and Andrew Lan. 2024. Improving Socratic Question Generation using Data Augmentation and Preference Optimization. In *Proceedings of the 19th Workshop on Innovative Use of NLP for Building Educational Applications (BEA 2024)*. Association for Computational Linguistics, Mexico City, Mexico, 108–118.

[3] Deborah L. Butler and Philip H. Winne. 1995. Feedback and Self-Regulated Learning: A Theoretical Synthesis. *Review of Educational Research* 65, 3 (1995), 245–281. https://doi.org/10.3102/00346543065003245

[4] Huayu Chen, Guande He, Lifan Yuan, Ganqu Cui, Hang Su, and Jun Zhu. 2024. Noise Contrastive Alignment of Language Models with Explicit Rewards. In *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada.*

[5] Badhan Chandra Das, M. Hadi Amini, and Yanzhao Wu. 2025. Security and Privacy Challenges of Large Language Models: A Survey. *ACM Comput. Surv.* 57, 6, Article 152 (Feb. 2025), 39 pages. https://doi.org/10.1145/3712001

[6] Paul Denny, James Prather, Brett A. Becker, James Finnie-Ansley, Arto Hellas, Juho Leinonen, Andrew Luxton-Reilly, Brent N. Reeves, Eddie Antonio Santos, and Sami Sarsa. 2024. Computing Education in the Era of Generative AI. *Commun. ACM* 67, 2 (Jan 2024), 56–67. https://doi.org/10.1145/3624720

[7] Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, et al. 2024. The Llama 3 Herd of Models. arXiv:2407.21783 [cs.AI] https://arxiv.org/abs/2407.21783

[8] Rodrigo Duran, Albina Zavgorodniaia, and Juha Sorva. 2022. Cognitive Load Theory in Computing Education Research: A Review. *ACM Trans. Comput. Educ.* 22, 4, Article 40 (sep 2022), 27 pages. https://doi.org/10.1145/3483843

[9] John Hattie and Helen Timperley. 2007. The Power of Feedback. *Review of Educational Research* 77, 1 (2007), 81–112. https://doi.org/10.3102/003465430298487

[10] Arto Hellas, Juho Leinonen, and Leo Leppänen. 2024. Experiences from Integrating Large Language Model Chatbots into the Classroom. In *Proceedings of the 2024 on ACM Virtual Global Computing Education Conference V. 1* (Virtual Event, NC, USA) *(SIGCSE Virtual 2024)*. Association for Computing Machinery, New York, NY, USA, 46–52. https://doi.org/10.1145/3649165.3690101

[11] Arto Hellas, Juho Leinonen, Sami Sarsa, Charles Koutcheme, Lilja Kujanpää, and Juha Sorva. 2023. Exploring the Responses of Large Language Models to Beginner Programmers' Help Requests. In *Proceedings of the 2023 ACM Conference on International Computing Education Research - Volume 1* (Chicago, IL, USA) *(ICER '23)*. Association for Computing Machinery, New York, NY, USA, 93–105. https://doi.org/10.1145/3568813.3600139

[12] Yann Hicke, Anmol Agarwal, Qianou Ma, and Paul Denny. 2023. AI-TA: Towards an Intelligent Question-Answer Teaching Assistant using Open-Source LLMs. arXiv:2311.02775 [cs.LG]

[13] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022. LoRA: Low-Rank Adaptation of Large Language Models. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022.*

[14] Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Kai Dang, et al. 2024. Qwen2. 5-Coder Technical Report. *arXiv preprint arXiv:2409.12186* (2024).

[15] Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2022. Large language models are zero-shot reasoners. In *Proceedings of the 36th International Conference on Neural Information Processing Systems* (New Orleans, LA, USA) *(NIPS '22)*. Curran Associates Inc., Red Hook, NY, USA.

[16] Nachiket Kotalwar, Alkis Gotovos, and Adish Singla. 2024. Hints-In-Browser: Benchmarking Language Models for Programming Feedback Generation. In *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024.*

[17] Charles Koutcheme, Nicola Dainese, and Arto Hellas. 2024. Using Program Repair as a Proxy for Language Models' Feedback Ability in Programming Education. In *Proceedings of the 19th Workshop on Innovative Use of NLP for Building Educational Applications (BEA 2024)*. Association for Computational Linguistics, Mexico City, Mexico, 165–181.

[18] 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 V. 1* (Pittsburgh, PA, USA) *(SIGCSETS 2025)*. Association for Computing Machinery, New York, NY, USA, 624–630. https://doi.org/10.1145/3641554.3701791

[19] Charles Koutcheme, Nicola Dainese, Sami Sarsa, Arto Hellas, Juho Leinonen, and Paul Denny. 2024. Open Source Language Models Can Provide Feedback: Evaluating LLMs' Ability to Help Students Using GPT-4-As-A-Judge. In *Proceedings of the 2024 Innovation and Technology in Computer Science Education, Volume 1* (Milan, Italy) *(ITICSE '24)*. https://doi.org/10.1145/3649217.3653612

[20] Rongxin Liu, Carter Zenke, Charlie Liu, Andrew Holmes, Patrick Thornton, and David J. Malan. 2024. Teaching CS50 with AI: Leveraging Generative Artificial Intelligence in Computer Science Education. In *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 2* (Portland, OR, USA) *(SIGCSE 2024)*. Association for Computing Machinery, New York, NY, USA, 1927. https://doi.org/10.1145/3626253.3635427

[21] Rongxin Liu, Julianna Zhao, Benjamin Xu, Christopher Perez, Yuliia Zhukovets, and David J. Malan. 2025. Improving AI in CS50: Leveraging Human Feedback for Better Learning. In *Proceedings of the 56th ACM Technical Symposium on Computer Science Education V. 1* (Pittsburgh, PA, USA) *(SIGCSETS 2025)*. Association for Computing Machinery, New York, NY, USA, 715–721. https://doi.org/10.1145/3641554.3701945

[22] Dominic Lohr, Hieke Keuning, and Natalie Kiesler. 2025. You're (Not) My Type—Can LLMs Generate Feedback of Specific Types for Introductory Programming Tasks? *Journal of Computer Assisted Learning* 41, 1 (2025), 2025. https://doi.org/10.1111/jcal.13107

[23] Ali Malik, Mike Wu, Vrinda Vasavada, Jinpeng Song, Madison Coots, John Mitchell, Noah Goodman, and Chris Piech. 2021. Generative Grading: Near Human-level Accuracy for Automated Feedback on Richly Structured Problems. In *Proceedings of the 14th Educational Data Mining conference.*

[24] Burt L. Monroe, Michael P. Colaresi, and Kevin M. Quinn. 2017. Fightin' Words: Lexical Feature Selection and Evaluation for Identifying the Content of Political Conflict. *Political Analysis* 16, 4 (2017), 372–403. https://doi.org/10.1093/pan/mpn018

[25] Tung Phung, Victor-Alexandru Pădurean, Anjali Singh, Christopher Brooks, José Cambronero, Sumit Gulwani, Adish Singla, and Gustavo Soares. 2024. Automating Human Tutor-Style Programming Feedback: Leveraging GPT-4 Tutor Model for Hint Generation and GPT-3.5 Student Model for Hint Validation. In *Proceedings of the 14th Learning Analytics and Knowledge Conference* (Kyoto, Japan) *(LAK '24)*. Association for Computing Machinery, New York, NY, USA, 12–23. https://doi.org/10.1145/3636555.3636846

[26] Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. 2023. Direct Preference Optimization: Your Language Model is Secretly a Reward Model. In *Advances in Neural Information Processing Systems*, Vol. 36. Curran Associates, Inc., 53728–53741.

[27] Alexander Scarlatos, Digory Smith, Simon Woodhead, and Andrew Lan. 2024. *Improving the Validity of Automatically Generated Feedback via Reinforcement Learning*. Springer Nature Switzerland, 280–294. https://doi.org/10.1007/978-3-031-64302-6_20

[28] Valerie J. Shute. 2008. Focus on Formative Feedback. *Review of Educational Research* 78, 1 (2008), 153–189. https://doi.org/10.3102/0034654307313795

[29] Annapurna Vadaparty, Daniel Zingaro, David H. Smith IV, Mounika Padala, Christine Alvarado, Jamie Gorson Benario, and Leo Porter. 2024. CS1-LLM: Integrating LLMs into CS1 Instruction. In *Proceedings of the 2024 on Innovation and Technology in Computer Science Education V. 1* (Milan, Italy) *(ITiCSE 2024)*. Association for Computing Machinery, New York, NY, USA, 297–303. https://doi.org/10.1145/3649217.3653584

[30] Arto Vihavainen, Jonne Airaksinen, and Christopher Watson. 2014. A systematic review of approaches for teaching introductory programming and their influence on success. In *Proceedings of the tenth annual conference on International computing education research*. 19–26.

[31] Sierra Wang, John Mitchell, and Chris Piech. 2024. A Large Scale RCT on Effective Error Messages in CS1. In *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1* (Portland, OR, USA) *(SIGCSE 2024)*. Association for Computing Machinery, New York, NY, USA, 1395–1401. https://doi.org/10.1145/3626252.3630764

[32] Juliette Woodrow, Sanmi Koyejo, and Chris Piech. 2025. Improving Generative AI Student Feedback: Direct Preference Optimization with Teachers in the Loop. https://juliettewoodrow.github.io/paper-hosting/dpo_feedback.pdf.

[33] Mike Wu, Noah Goodman, Chris Piech, and Chelsea Finn. 2021. Prototransformer: A Meta-Learning Approach to Providing Student Feedback. arXiv:2107.14035 [cs.CY] https://arxiv.org/abs/2107.14035

[34] Zezhu Yu, Suqing Liu, Paul Denny, Andreas Bergen, and Michael Liut. 2025. Integrating Small Language Models with Retrieval-Augmented Generation in Computing Education: Key Takeaways, Setup, and Practical Insights. In *Proceedings of the 56th ACM Technical Symposium on Computer Science Education V. 1* (Pittsburgh, PA, USA) *(SIGCSETS 2025)*. Association for Computing Machinery, New York, NY, USA, 1302–1308. https://doi.org/10.1145/3641554.3701844

[35] Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, et al. 2023. Judging LLM-as-a-judge with MT-bench and Chatbot Arena. In *Proceedings of the 37th International Conference on Neural Information Processing Systems* (New Orleans, LA, USA) *(NIPS '23)*. Curran Associates Inc., Red Hook, NY, USA, Article 2020, 29 pages.

[36] Chunting Zhou, Pengfei Liu, Puxin Xu, Srini Iyer, Jiao Sun, et al. 2023. LIMA: Less Is More for Alignment. arXiv:2305.11206 [cs.CL]