

Reinforcement Learning for Programming Feedback: Aligning Small Language Models Without Human Preferences

Charles Koutcheme^{1,*}, Nicola Dainese¹ and Arto Hellas¹

¹Aalto University, Espoo, Finland

Abstract

Providing students with timely and effective feedback remains a critical challenge in programming education. Locally deployed Small Language Models (SLMs) offer a cost-effective solution that enables educators to generate feedback while avoiding third-party reliance and privacy concerns associated with Large Language Models (LLMs). However, SLMs often produce misleading or inaccurate feedback, limiting their practical use. This paper presents a fully automated reinforcement learning framework for aligning SLMs to generate high-quality programming feedback without any human-labelled examples or preference annotations. Our approach transfers the feedback capabilities of powerful LLMs (“teacher models”) to smaller, low-resource models (“student models”) that can run locally on consumer hardware, with the optional assistance of medium-sized “assistant” models. The framework supports two configurations: an off-policy setup that uses assistant model generations to bootstrap alignment and a lightweight online on-policy variant that trains directly on student model outputs. We evaluate both approaches by fine-tuning two SLMs on a real-world dataset of CS1 programming submissions collected across semesters. Our experiments simulate realistic deployment scenarios, training on data from past semesters and evaluating on future ones. Results show that both methods significantly improve feedback quality and generalize across new course offerings. We provide practical considerations for aligning SLMs in educational settings and outline a promising direction for future work. Our code is made available on GitHub.

Code:  github.com/KoutchemeCharles/rlpf

Keywords

Language Models, Programming Feedback, Computing Education, Reinforcement Learning,

1. Introduction

Learning to program is challenging for many. These challenges can be somewhat alleviated with improved teaching practice [1, 2]. A key part of this is providing feedback, which should be timely and accurate [3, 4, 5]. Large Language Models (LLMs) have shown exceptional success in that task [6, 7], leading to their growing adoption in classrooms [8, 9, 10, 11, 12]. However, relying on third-party services that provide access to LLMs can introduce cost obstacles and scalability issues [13]. These constraints are driving a growing shift towards using smaller, open-source models [14], which can be deployed locally [15, 16] to reduce costs and provide educators greater control over their students’ data.

Although Small Language Models (SLMs) alleviate these issues due to the ability to run them locally [15], their feedback quality often falls short of LLMs [17], posing significant challenges in real-world applications. In particular, SLMs tend to generate more misleading feedback [14, 17], including hallucinations and irrelevant suggestions. Such shortcomings can confuse students and hinder learning [18].


Reinforcement Learning (RL) has emerged as a promising approach for aligning language models to generate pedagogically meaningful programming support [19]. However, existing reinforcement learning methods for programming feedback generation rely heavily on human supervision, typically in the form of human-written examples [20] or preference annotations [21]. This dependency hinders improvements in contexts where data or annotators are unavailable.

CSEDM’25: 9th Educational Data Mining in Computer Science Education Workshop, July, 2025, Palermo, Italy

*Corresponding author.

 charles.koutcheme@aalto.fi (C. Koutcheme); nicola.dainese@aalto.fi (N. Dainese); arto.hellas@aalto.fi (A. Hellas)

 <https://koutche.me/> (C. Koutcheme)

 0000-0002-0877-7063 (C. Koutcheme); 0000-0001-7116-9338 (N. Dainese); <https://orcid.org/0000-0001-6502-209X> (A. Hellas)



© 2025 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

In this paper, we explore aligning small models for programming feedback without human annotations or preference labels. Our approach uses RL to transfer the feedback abilities from a teacher LLM to a smaller, locally deployable student model, optionally using medium-sized assistant models to bootstrap alignment. We implement and compare two fully automated training configurations: an off-policy setup (TASAP), which builds on prior work using assistant generations [22]; and a novel lightweight online on-policy method (OSAP), where the model trains directly on its own feedback.

Our evaluation focuses on feedback generation for explanations of students’ mistakes [23] and Socratic hints [24], using two SLMs (SmolLM-V2-1.7B [25] and Llama-3.1-1B [26]) fine-tuned on real student submissions from the FalconCode dataset [27]. We use this dataset to simulate a realistic deployment scenario by training on one semester and evaluating on the next. We also study a continual learning setting where models are refined incrementally as new data becomes available.

Our results show that both configurations significantly improve feedback quality on new student submissions. We also reflect on methodological challenges when training small models for educational feedback and highlight a promising direction for future work. Our contributions are the following:

- We introduce a reinforcement learning framework for aligning small language models for programming feedback, without relying on human-labelled preferences or human-annotated feedback.
- We implement an off-policy method using assistant models (TASAP) and introduce a novel online on-policy variant (OSAP).
- We evaluate both methods on a real-world, semester-split dataset, and show that they substantially improve feedback quality on new students’ submissions, including in a continual learning setup.

2. Background and Related Work

2.1. Learning from Preferences

Reinforcement Learning With Human Feedback Recent advancements in fine-tuning techniques have significantly improved the performance of small language models on downstream tasks. While Supervised Fine-Tuning (SFT) remains a widely used approach to improve language models’ generations [28], it is limited in its ability to align such models with complex human preferences and objectives [29]. Reinforcement Learning with Human Feedback (RLHF) addresses this limitation by training reward models on ranked human preferences — for example, generation A is better than generation B — and has contributed to the success of large language models such as GPT-4 [30].

LLMs-as-judges. Because of their strong performance, such models have also been used as “judges” to evaluate outputs from smaller models [31, 14]. The growing use of LLMs-as-judges has progressively reduced the need for human annotators in RLHF pipelines [32], creating a shift towards Reinforcement Learning From AI Feedback (RLAIF) [33], where AI models themselves are used to supervise other models’ preference-based training.

Direct Preference Optimization. Recent RLHF and RLAIF approaches have predominantly relied on offline preference alignment methods such as Direct Preference Optimisation (DPO) [34]. DPO simplify the classical three-step pipeline by directly optimising language models on prior collected preference data, removing the need to train a separate reward model [35].

Parameter-Efficient Fine-Tuning. In parallel, Parameter-Efficient Fine-Tuning (PEFT) techniques [36], such as Low-Rank Adapters (LoRA) [37], have made fine-tuning more accessible by significantly reducing computational and memory requirements. These techniques have enabled the practical application of preference alignment to smaller, resource-constrained models.

2.2. Improving Small Language Models for Programming Feedback

Existing reinforcement learning approaches. Fine-tuning small language models (SLMs) for programming education has become an increasingly active area of research in AI in Education. While early work primarily focused on generating program repairs to correct student code [38, 39], more recent approaches explore reinforcement learning (RL) and PEFT methods to fine-tune small models to support students’ learning how to program. In all setups, a common challenge is obtaining high-quality preference pairs to guide the learning process.

Some approaches rely on human annotations paired with synthetic examples. For instance, Kumar et al. use GPT-4 to generate low-quality samples, paired with human-written Socratic questions, to train a LLaMA model for educational dialogue [20]. Other approaches leverage naturally occurring preference signals. Hicke et al. use TA edits to forum posts as implicit preferences [19].

Towards Reinforcement Learning with AI Feedback. While promising, these methods rely on human supervision or access to structured educational data, which limits their scalability to new contexts. Kotalwar et al. take a step toward automation by using GPT-4 to generate explanations and hints, training a small model via supervised fine-tuning alone [40]. However, whether preference-based techniques can further improve such models without human annotation remains underexplored.

Recent studies highlight the promise of LLMs-as-judges in evaluating feedback quality, with models such as GPT-4o-mini and Llama-3.1-70B producing high-quality judgments [14, 17]. These advances motivate our work to adapt RLAIIF for programming feedback.

Closest to our work in another domain is Scarlatos et al. [22], who use a combination of human-written feedback, LLM-generated feedback, and AI preferences to train an 8B LLaMA model with PEFT for multiple-choice math feedback, using Direct Preference Optimisation. While both studies rely on RLAIIF, our approach differs by integrating such techniques within a distillation framework that addresses specific programming feedback challenges, notably, the lack of human-annotated data, the vast space of possible student mistakes, and the need for highly contextualised recommendations.

Moreover, our work differs from all prior attempts by also integrating online learning algorithms [41], where language models improve continuously with their own generated responses.

3. Methods

Here, we present our two approaches for improving small language models’ programming feedback. Before presenting the training methods, we formalize the task and outline our assumptions.

3.1. Task and Assumptions

Task. Our primary objective is to fine-tune a small, resource-efficient, instruction-tuned language model π_θ (the *student* LM) to generate two interrelated types of feedback [42, 40, 24]: an explanation \mathcal{E} , which identifies and describes a bug in a student’s program, and a single next-step hint \mathcal{H} , which guides the student toward resolving the identified bug without revealing the solution.

While our method can be adapted to other types of feedback [3], we illustrate its effectiveness with explanations and hints as these two types of feedback play an important role in supporting students learning programming.

Quality attributes. To ensure the feedback supports effective learning, it must adhere to specific quality attributes identified in prior works. First, the generated explanation must be accurate, selective, and clear [14]. The explanation is considered accurate (\mathcal{E}_A) if it correctly identifies and mentions the first existing issue in the student program. It is considered selective (\mathcal{E}_S) when it focuses exclusively on one issue in the code (whether the issue is correct or not) and avoids discussing any unrelated or non-existent bugs. Finally, the explanation should be clear (\mathcal{E}_{Cle}), meaning it is easy to understand, concise, and presented in a readable format.

Second, the generated hint must be correct, informative, concealed, and clear [42]. A hint is considered correct (\mathcal{H}_C) if it provides accurate information to resolve issues in the buggy program. It is deemed informative (\mathcal{H}_I) if it offers valuable insights to help the learner resolve the bug effectively.

The hint should also remain concealed (\mathcal{H}_{Con}) by avoiding the direct revelation of the solution, to reason through the process of implementing the fix. Lastly, the hint must be clear (\mathcal{H}_{Cle}), ensuring that it is easy to understand and devoid of unnecessary complexity. The student language model, π_θ , will be optimized to consistently meet these quality attributes in its generations.

Generation methodology. Following prior work [42, 40], feedback \mathcal{F} is always generated using a chain-of-thought approach that prompts language models to generate the explanation \mathcal{E} (the “thought”) followed by the hint \mathcal{H} ; This strategy ensures hints are grounded in accurate explanations.

Assumptions. To reach our objective, we consider a training dataset $\mathcal{D} = \{(d^i, s^i)\}_{i=1}^N$ consisting of N pairs of problem descriptions d^i and incorrect student programs s^i .

We also assume to have access to a *teacher* LLM, π_τ , accessible via an online API (e.g., GPT-4o-mini via OpenAI API). We also suppose having access to a set of A medium-sized *assistant* LMs π_{α_j} , $j \in \{1, \dots, A\}$. The teacher model is presumed to generate high-quality feedback, while the assistant models perform well but with lower quality, and the student model may initially perform poorly.

3.2. Supervised Fine-tuning

Given the lack of human annotations, following Koltawar et al. [40], a natural first step in improving our small language model is to apply Supervised Fine-Tuning (SFT), that is, training the student model on teacher-generated feedback for all incorrect programs in the training set using the negative log-likelihood (NLL) loss. This yields a model π_{sft} . We generate such feedback $\mathcal{F}_{\pi_\tau}^i$ using greedy decoding. Figure 2 (Appendix B) shows our prompt.

SFT represents the simplest form of distillation [43], where the student directly mimics the teacher’s outputs. However, SFT alone risks overfitting, especially when training data is limited, and does not allow language models to understand what constitutes high-quality responses.

3.3. Learning From Feedback Preferences

In this paper, we propose to apply preference-based optimisation techniques on top of the SFT-trained small language models to refine their abilities to generate high-quality feedback. Unlike RLHF setups that rely on human preference labels [21], we generate preferences automatically. We compare two configurations that vary in how feedback examples are generated and how the student model is updated.

In both setups, we use the teacher model to score and rank the generated feedback using a rubric-based process before optimizing the student model via an appropriate preference alignment algorithm.

3.3.1. Teacher-Assistant-Student Alignment Pipeline (TASAP)

Our first approach, the Teacher-Assistant-Student Alignment Pipeline (TASAP), follows similar offline off-policy preference alignment strategies underpinning the success of many language models [32, 44]. To apply such methods in our context, we need to construct a preference dataset with feedbacks f , $\mathcal{D}_p = \{(d^i, s^i, f_w), (d^i, s^i, f_l)\}_{i=1}^M$, where f_w (the “winning” feedback) is ranked higher than f_l (the “loosing” feedback) based on a quality criterion.

Step 1: Data collection. For each incorrect program, we sample three feedback, each one from the assistant models $\mathcal{F}_{\pi_{\alpha_j}}^i$ using greedy-decoding following prior work [14, 31]. We also reuse the feedback generated by the teacher language model $\mathcal{F}_{\pi_\tau}^i$ during the supervised fine-tuning step.

Step 2: Judging and scoring generations. Then, we use our teacher π_τ as a judge [31] to grade all four generated feedback (independently against a rubric based on our predefined quality criteria: \mathcal{E}_A , \mathcal{E}_S , ..., \mathcal{H}_C , \mathcal{H}_I , \mathcal{H}_{Con} , and \mathcal{H}_{Cle} , assigning each criterium a binary value of either 0 (false) or 1 (true) [22]. Following Koutchme et al. [14], our prompt for judging feedback (see Figure 3, Appendix B) asks the teacher LM π_τ to use its own generated feedback as *ground truth* to evaluate the newly provided one. This reference grading strategy [31] ensures the student generations remain aligned with the teacher, reduces variability in judgments, and ensures the preference dataset is free of noise [45]. Using the grading values, we assign each feedback an overall quality score [22, 32] using a weighted sum:

$$\mathcal{S}_f = 0.20 \cdot \mathcal{E}_A + 0.15 \cdot \mathcal{E}_S + 0.10 \cdot \mathcal{E}_C + 0.20 \cdot \mathcal{H}_C + 0.15 \cdot \mathcal{H}_I + 0.10 \cdot \mathcal{H}_{Con} + 0.10 \cdot \mathcal{H}_{Cle}$$

where the resulting score \mathcal{S} is also bounded between 0 and 1. Our scoring function prioritizes explanation correctness and hint accuracy to ensure feedback is factual. We then consider explanation selectivity and hint informativeness to discourage the generation of irrelevant or hallucinated information. Attributes like clarity and concealment are considered last, as they are secondary to the validity of the feedback (the scoring function can be adapted by teachers to match their needs).

Step 3 - Preference dataset creation. Using the four feedback obtained, three from the SFT model π_{sft} , one by the teacher π_τ , for all given incorrect programs s^i , we add to our preference dataset \mathcal{D}_p all possible feedback pairs (f_w, f_l) where f_w score is better than f_l score ($\mathcal{S}_{f_w} > \mathcal{S}_{f_l}$).

Step 4 - Optimization. Using the resulting preference dataset, we train our language model using the DPO loss function [34]:

$$\mathcal{L}_{DPO}(\pi_\theta; \pi_{sft}) = -\mathbb{E}_{(s^i, d^i, f_w, f_l) \sim \mathcal{D}_p} \left[\log \sigma \left(\beta \log \frac{\pi_\theta(f_w | s^i, d^i)}{\pi_{sft}(f_w | s^i, d^i)} - \beta \log \frac{\pi_\theta(f_l | s^i, d^i)}{\pi_{sft}(f_l | s^i, d^i)} \right) \right] \quad (1)$$

where σ is the logistic function, π_θ is the policy being optimized (i.e., the model during training), π_{sft} is the reference policy (i.e., the frozen model before training), and β is a regularization parameter that controls the deviation of the trained from the reference policy. A higher β keeps the trained model closer to the reference policy. Intuitively, this formulation penalizes the model based on how much it “prefers” the lower-quality (losing) feedback over the higher-quality (winning) feedback, which results in gradually increasing the probability of generating high-quality outputs.

3.3.2. Online Student Alignment Pipeline (OSAP)

Our second approach, Online Student Alignment Pipeline (OSAP), is an online on-policy variant of TASAP based on Direct Language Model Alignment from Online AI Feedback [41]. Compared to offline approaches, online training continuously updates a language model based on its own generations, potentially reducing common issues associated with using static preference datasets, such as distribution shift [34] and overfitting [46].

Starting from the supervised fine-tuned model π_{sft} , OSAP integrates the sampling, data collection, and optimization steps of the TASAP pipeline within a single optimization loop. At each iteration:

- (a) Instead of sampling from assistant models, we sample two generations $f_1, f_2 \sim \pi_{sft}(s^i, d^i)$ from our language model using multinomial sampling with an arbitrary temperature of 0.3 [41].
- (b) We use our teacher model π_τ to independently judge and score each generation to determine the winning f_w and losing feedback f_l , before updating the model parameters based on the resulting preference ordering using the original DPO loss function (see equation 1). If both generations obtain the same score, we default to syntactic distance measures, and we select the feedback having the highest ROUGE score [47] with the teacher feedback [19].

3.4. Adaptation to New Student Data

In real-life scenarios, student programming submissions are collected by course offerings (e.g. semester by semester) and accumulate and even somewhat change over time [48]. To take this scenario into account, we need to study effectively how each of the two preference-based alignment strategies, TASAP and OSAP, can be applied when additional training data is introduced. We consider the task of refining a model already trained on an initial dataset \mathcal{D}_1 , using a new semester of data $\mathcal{D}_2 = \{(d^l, s^l)\}_{l=1}^M$.

TASAP : We perform steps 1 to 4 of the TASAP pipeline on the new dataset of students’ incorrect programs \mathcal{D}^2 to obtain a second preference dataset \mathcal{D}_p^2 . Training the first model exclusively on this new dataset might induce a situation of catastrophic forgetting [49], where the student model loses some of the knowledge it acquired when trained on \mathcal{D}_p^1 . To mitigate this issue, we initialize the weights of our model to the supervised fine-tuning version (see section 3.2) of the first semester (i.e., π_{sft}) and train this model using the IPO loss on the combined $\mathcal{D}_p^1 \cup \mathcal{D}_p^2$. Our choice of not repeating the supervised fine-tuning step on the combined dataset, and instead, starting from π_{sft} is motivated by our tentative to mitigate overfitting risks.

OSAP : For OSAP, we continue the training pipeline directly¹ from $\pi_{\mathcal{D}_p}$ (i.e., the OSAP model trained on the first semester), using the problem description and incorrect programs from $\mathcal{D}_p^1 \cup \mathcal{D}_p^2$. This strategy thus reflects a true continual learning setup and most benefits from new data.

We note that both techniques shown can be applied continuously, for instance, for refining the model trained on two semesters of data using a third one.

4. Experiments

In this section, we present our experiments, aiming to answer the following research question:

(RQ) *How effective are TASAP and OSAP in improving the feedback quality of small language models when trained and evaluated across semesters of the same introductory programming course?*

4.1. Dataset

We perform our experiments using FalconCode [27], a large and comprehensive publicly available dataset containing real-life CS1 students’ solutions to Python programming exercises. Beyond its substantial scale, this dataset distinguishes itself through free-form assignments, enabling a broader evaluation of language models’ abilities to generate feedback.

Preprocessing. The FalconCode dataset is split over three subsets (three semesters of data). Within each subset, we select all unique incorrect programs from all students’ last submitted solutions for all assignments automatically evaluated with unit tests [50]. Uniqueness is determined via AST normalization². While we acknowledge that this selection may not fully capture the range of difficulties students encounter during their attempts, it aligns with the idea that a student’s last attempt often reflects their improved understanding of the problem. Thus, our setup can be viewed as providing feedback to students as a last resort for elements they may not have grasped.

We leverage the first and second semesters for training and iterative refinement, respectively, and the last semester for testing. To ensure our setup evaluates our models’ generalization abilities, we filter out from the test set the programs in the first two semesters having similar normalised AST representations [51]. This results in three splits with 826, 690, and 693 incorrect programs (s_i) from 62, 44, and 62 assignments (d_j), respectively.

¹In practice, we also need to generate feedback using the teacher model on the new semester of data \mathcal{D}_p^2 to allow the fall back to a syntactic distance measure comparison.

²Including variable renaming.

4.2. Models

To answer our research questions, we fine-tune two small language models, SmolLM-V2-1.7B [25] and Llama-3.1-1B [26], using GPT-4o-mini [52] as the teacher. We chose these two student models for their strong performance on small-model benchmarks, while GPT-4o-mini has been shown to produce high-quality programming feedback [17].

Baseline. As a baseline, we use the models trained using Supervised Finetuning on each teacher-generated data following Kotalwar et al. [40].

Versions. We train each of our models (on FalconCode) using our two proposed approaches: TASAP and OSAP. For each approach, we train a first version (TASAP-1 and OSAP-1) on FalconCode first semester. We train second versions of our models using the adaptation to new student data strategy (see section 3.4) with both FalconCode first and second semesters.

Assistant models. For TASAP, we leverage three assistant language models: Mistral-Nemo-12B [44], Llama-3.1-8B [26], and Qwen-2.5-3B [53]. We chose these models to ensure diversity across model families, sizes, and performance [32].

Parameter Efficient Finetuning To take into account educators’ limited access to computational resources, we train our SFT and TASAP models (as well as the baselines introduced below) with Low-Rank Adapters (LoRA) [37], a parameter-efficient fine-tuning method that reduces memory requirements by freezing the base model and adding a small number of trainable parameters called adapters [36]. These adapters can be removed to restore the base model’s original capabilities.

4.3. Automated Evaluations: LLMs-as-feedback-judges

Manually evaluating all models on our datasets would require substantial effort, even on a subset of generations. Instead, we leverage LLMs-as-judges once again for our final evaluation [31]. However, rather than relying on a single model for this task, following Verga et al. [54], we use a panel of three strong LLMs: Llama-3.3-70B [26], GPT-4o-mini, [26], and Gemini-2.0-flash [55]. Earlier versions of the GPT-4o and the Llama-3 family have been used extensively as judges [56], also in programming context [17, 14], and Gemini has recently demonstrated comparable performance to GPT-4o-mini on multiple benchmarks. While GPT-4o-mini and Gemini-2.0-flash are lighter versions of their full-size counterparts, they remain strong judges for programming feedback. For instance, GPT-4o-mini has been shown to perform on par with GPT-4o for evaluating feedback quality [17]. Moreover, Verga et al. [54] demonstrate that ensembles of smaller LLMs of different families outperform single large models, particularly by mitigating individual model biases.

Evaluation prompting strategy. For each feedback \mathcal{F} generated on the test set, we prompt all judges (see Figure 4, Appendix B) to provide binary decisions across all quality criteria. We obtain the final verdict using a strict unanimity policy: a criterion is marked correct only if all judges agree. While this method does not provide absolute performance guarantees, as discussed in our Limitations of Work, it offers a consistent, scalable, and reliable strategy for comparing the relative effectiveness of different training approaches.

Human validation. Following Scarlatos et al. [22], we conduct a small-scale analysis over a subset of language model generations to validate the use of LLM-as-judges and provide insights into potential evaluation errors.

4.4. Experiment details

We fine-tune our models using the HuggingFace TRL library, following hyperparameters recommended in prior work. For SFT, we use a learning rate of $1e-4$ [34]; for TASAP and OSAP, we set $\beta = 0.25$ [41] and use learning rates of $1e-5$ and $1e-6$, respectively. Batch sizes are 8 for SFT and TASAP, and 16 for OSAP. TASAP-2 and OSAP-2 reuse these settings and repeat the training process as described in Section 3.4. We apply LoRA with $\alpha = 64$ and rank $r = 32$ [37], train each model for up to 3 epochs, and select checkpoints based on lowest validation loss. All other hyper-parameters remain at default values. Full experimental details and prompts are available in our code base. All training was performed on Nvidia Tesla V100 GPUs (32GB RAM) via *Triton*, our institution’s research cluster.

5. Results

5.1. Main results

Table 1 shows the results of our experiments. We can observe the following.

Table 1

Percentage of incorrect programs with a given feedback quality. We bold the best results per model family (Llama-3.1-1B or Smol2-1.7B). Model legend: BASE: untrained model, SFT: Supervised Fine-tuned model, TASAP(-2): Teacher-Assistant-Student Alignment Pipeline (trained on 2 semesters), OSAP(-2): Online Student Alignment Pipeline (trained on 2 semesters). QWEN: Qwen-2.5-3B, LLAMA: Llama-3.1-8B, NEMO: Mistral-Nemo-12B, MINI: GPT-4o-mini. Explanation (\mathcal{E}) criteria: \mathcal{E}_A : accuracy, \mathcal{E}_S : selectivity, \mathcal{E}_{Cle} : clarity. Hint criteria (\mathcal{H}): \mathcal{H}_C : correctness, \mathcal{H}_I : informativeness, \mathcal{H}_{Con} : concealment, \mathcal{H}_{Cle} : clarity.

Llama-3.1-1B (Student)								Smol2-1.7B (Student)							
Model	\mathcal{E}_A	\mathcal{E}_S	\mathcal{E}_{Cle}	\mathcal{H}_C	\mathcal{H}_I	\mathcal{H}_{Con}	\mathcal{H}_{Cle}	Model	\mathcal{E}_A	\mathcal{E}_S	\mathcal{E}_{Cle}	\mathcal{H}_C	\mathcal{H}_I	\mathcal{H}_{Con}	\mathcal{H}_{Cle}
BASE	20.7	16.3	46.6	27.4	7.5	81.2	62.2	BASE	29.5	43.9	48.0	33.9	9.8	86.4	58.4
SFT	60.6	55.8	55.7	67.9	15.2	98.3	81.7	SFT	53.6	67.4	52.2	61.5	15.8	97.6	77.1
OSAP	63.6	58.1	57.7	71.6	13.1	98.6	83.3	OSAP	60.8	71.6	57.1	66.7	16.7	98.2	81.6
TASAP	68.5	61.6	55.1	76.8	16.3	99.2	86.4	TASAP	66.7	74.5	63.4	72.0	20.8	98.9	85.8
TASAP-2	68.2	64.1	57.5	76.5	15.9	99.0	86.5	TASAP-2	68.2	76.1	65.1	75.9	26.6	98.6	85.7
OSAP-2	69.5	64.3	53.0	77.8	14.2	98.7	85.4	OSAP-2	73.0	78.8	55.4	80.6	20.3	99.4	88.2
MINI (TEACHER) AND QWEN (ASSISTANT)								LLAMA AND NEMO (ASSISTANTS)							
MINI	98.7	98.0	73.8	98.9	24.9	100.0	99.2	Llama	82.6	75.5	74.5	87.1	20.4	99.9	93.0
QWEN	70.9	66.2	75.0	72.9	19.4	96.1	88.5	NEMO	90.3	87.2	74.7	92.7	16.9	99.6	97.0

We observe that both TASAP and OSAP generally improve performance over the supervised fine-tuned baselines. Interestingly, Llama TASAP (respectively OSAP) achieves performance comparable to Smol OSAP (respectively TASAP), despite the more substantial performance difference between the respective supervised fine-tuned base models.

Tracing this observation backwards, we note that although the Smol base model performs slightly better than the Llama base model (as expected, due to size differences), supervised fine-tuning benefits the Llama model more. We hypothesise that this may be due to a distribution shift: the Llama model’s answer distribution is closer to that of GPT-4o-mini, making further improvements easier [34]. Training with OSAP and TASAP may guide the model toward a more optimal solution space. Hieke et al. [19] have already highlighted that preference optimization exerts a regularising effect on supervised fine-tuning.

While TASAP generally outperforms OSAP across both language models, this performance gap narrows when training on additional data (e.g., from a subsequent semester). OSAP-2 performs comparably to TASAP-2 on Llama, and even outperforms TASAP-2 on Smol.

5.2. Small-scale human evaluation

Following Scarlatos et al. [22], we conduct a small-scale analysis of LLMs-as-judges performance in our setting. We arbitrarily selected a subset of 5 representative assignments in our dataset (see Table 1, Appendix A). For each assignment, we choose the student’s submitted incorrect solution which had the highest unit test score. Then, one author of the paper manually annotated the quality of the generations of the BASE, SFT, TASAP, and OSAP models for those 5 assignments for the two models, resulting in $4 \times 5 \times 2 = 40$ annotations with 7 criteria. Our analysis procedures follow prior work [17, 22], considering such manual annotations as ground truths and the LLM-as-judges ensemble result as predictions in 7 distinct binary classification problems (one per criteria). Table 2 shows the result of such annotations for various classification metrics.

Table 2

LLM-as-judges classification performance. We report various classification metrics. Legend: %PA: number of positive human annotations (out of 40) for each respective criteria. (\mathcal{E}) criteria: \mathcal{E}_A : accuracy, \mathcal{E}_S (selectivity), \mathcal{E}_{Cle} : Clarity. Hint criteria (\mathcal{H}): \mathcal{H}_C : correctness, \mathcal{H}_I : informativeness, \mathcal{H}_{Con} : Concealment, \mathcal{H}_{Cle} : Clarity.

#PA	\mathcal{E}_A 8/40	\mathcal{E}_S 6/40	\mathcal{E}_{Cle} 24/40	\mathcal{H}_C 7/40	\mathcal{H}_I 4/40	\mathcal{H}_{Con} 33/40	\mathcal{H}_{Cle} 34/40
f0.5	0.86	0.89	0.88	0.85	0.83	0.80	0.84
accuracy	0.82	0.82	0.88	0.80	0.90	0.85	0.78
precision	0.54	0.46	0.91	0.46	0.00	0.85	0.96
recall	0.88	1.00	0.88	0.86	0.00	1.00	0.76
f1	0.67	0.63	0.89	0.60	0.00	0.92	0.85
kappa	0.56	0.54	0.74	0.48	0.00	0.22	0.40

Leveraging LLMs as judges yields a minimum F0.5 score of 80% and a minimum accuracy of 80% for classifying model generations. For the selected assignments, most model outputs were neither accurate (\mathcal{E}_A) nor correct (\mathcal{H}_C), resulting in an imbalanced classification task. Similar to the main results in Table 1, we observe that language models struggle to generate informative hints. The LLM-judges ensemble did not classify any generation as containing an informative hint, and our human annotations identified only 4 out of 40 (10%). While LLMs are not perfect evaluators in general [57], our small-scale human analysis supports their utility in this context as a reasonable proxy for human judgment.

6. Discussion and Conclusion

In this paper, we presented a framework for improving small language models’ ability to provide feedback using Reinforcement Learning With AI Feedback. We proposed two approaches based on offline and online preference alignment methods and evaluated the methods performance using LLM-as-judges on a publicly available dataset of students’ Python programs. To summarize the answer to our research question, the proposed framework, including TASAP and OSAP methods, is effective in enhancing SLMs’ feedback capability within a course setting.

Practical educational implications. By utilizing and training fine-tuned models, educators can provide tailored guidance to their students in a timely manner without constantly relying on external APIs. Such small models can be effectively deployed using tools like WebLLM [58], even allowing inference on client devices with a compatible GPU. This reduces latency, ensures timely feedback, and eliminates the need to deploy custom inference services [40]. This can also give educators and learners higher control over the generated information and its use.

We do not aim to claim that pure data-driven approaches are the way to go. Ideally, when preference data can be collected and integrated by human TAs [21], such approaches should most likely be prioritised. However, in many instances, programming courses do not have human TAs write feedback to students or collect preference data, which limits how such prior work can be effectively used. Our work closes this gap.

Privacy and cost issues. We acknowledge that leveraging RLAIIF pipelines requires sending student data through external APIs for teacher model queries, which breaks privacy measures and might also incur initial costs. However, we also note that much of the existing work in programming education already relies on proprietary models (e.g. [40, 42]). Moreover, distilling the performance of remote queried models to smaller models run locally decreases long-term costs. For institutions with strict data privacy concerns, open-source LLMs (e.g. 4-bit quantized Llama-3.3-70B) could, given sufficient computational resources, also be hosted locally and used as teacher models.

Room for improvement. We note that our results can be further improved, for instance, by training with more data, leveraging several prompts for different feedback tasks simultaneously, and increasing LoRA rank (the parameter controlling how many parameters θ are updated during training). Programming educators and practitioners often have data readily available to them through the use of automated assessment systems.

The framework is versatile. We anticipate that this training procedure will generalise to more complex prompting strategies, for instance, leveraging program repairs [42] to produce better feedback, as the improvements stem from the framework’s alignment mechanisms rather than the specific prompt design. We could also have trained the model on several prompts for different types of feedback, using more data. Programming educators and practitioners often have data readily available to them through the use of automated assessment systems. The framework is adaptable; we recommend adopting the same setup as ours: using a teacher LLM to evaluate over one semester to understand what to expect.

Limitations of work. First, we conducted all experiments on a single dataset of Python programming submissions collected from one institution and did not explore whether our results hold in other contexts. Second, although our automated evaluation pipeline is robust, leveraging several leading large language models, no large-scale human analysis was performed. Third, our experiments were limited to two small models with around 1B parameters. While prior work suggests that performance improves with base model size [34], it remains to be seen whether the same trends hold when applying OSAP and TASAP to larger models. Fourth, importantly, we do not claim that OSAP and TASAP trained models produce feedback matching the exact reported scores (e.g., we do not assert that the models now generate “nearly perfect feedback”). Rather, the combination of a large dataset and the substantial performance margins allows us to confirm relative rankings with confidence, even when taking into account judgment error rates [14].

Future work. Future work will address these gaps by first conducting human evaluations to validate the usefulness of the feedback generated by our trained models. This will include qualitative surveys with both teachers and students to gain insights into their perspectives. We also plan to conduct small-scale A/B studies in real educational settings, comparing courses that use these locally deployed models as AI teaching assistants with those relying on larger models. These deployments will provide critical insights into small models’ impact on student learning, engagement, and overall educational outcomes.

Moving forward, we are studying ways to improve small language models’ programming feedback ability without relying on large language models. In particular, we believe the recent success of pure reinforcement learning methods such as Group Relative Preference Optimization [59] could also benefit programming education.

Declaration on Generative AI

During the preparation of this work, the author(s) used ChatGPT and Grammarly to: Grammar and spelling check. After using these tool(s)/service(s), the author(s) reviewed and edited the content as needed and take(s) full responsibility for the publication’s content.

References

- [1] A. Luxton-Reilly, Simon, I. Albluwi, B. A. Becker, M. Giannakos, A. N. Kumar, L. Ott, J. Paterson, M. J. Scott, J. Sheard, et al., Introductory programming: a systematic literature review, in: Proceedings companion of the 23rd annual ACM conference on innovation and technology in computer science education, 2018, pp. 55–106.
- [2] A. Vihavainen, J. Airaksinen, C. Watson, 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, 2014, pp. 19–26.
- [3] H. Keuning, J. Jeuring, B. Heeren, A systematic literature review of automated feedback generation for programming exercises, *ACM Transactions on Computing Education (TOCE)* 19 (2018) 1–43.
- [4] J. Hattie, H. Timperley, The power of feedback, *Review of educational research* 77 (2007) 81–112.
- [5] V. J. Shute, Focus on formative feedback, *Review of educational research* 78 (2008) 153–189.
- [6] D. Lohr, H. Keuning, N. Kiesler, You’re (not) my type—can llms generate feedback of specific types for introductory programming tasks?, *Journal of Computer Assisted Learning* 41 (2025) 2025. URL: <https://onlinelibrary.wiley.com/doi/10.1111/jcal.13107>. doi:10.1111/jcal.13107.
- [7] P. Denny, J. Prather, B. A. Becker, J. Finnie-Ansley, A. Hellas, J. Leinonen, A. Luxton-Reilly, B. N. Reeves, E. A. Santos, S. Sarsa, Computing education in the era of generative ai, *Commun. ACM* 67 (2024) 56–67. URL: <https://doi.org/10.1145/3624720>. doi:10.1145/3624720.
- [8] U. Z. Ahmed, S. Sahai, B. Leong, A. Karkare, 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, SIGCSETS 2025, Association for Computing Machinery, New York, NY, USA, 2025, p. 11–17. doi:10.1145/3641554.3701972.
- [9] R. Liu, C. Zenke, C. Liu, A. Holmes, P. Thornton, D. J. Malan, 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, SIGCSE 2024, Association for Computing Machinery, New York, NY, USA, 2024, p. 1927. URL: <https://doi.org/10.1145/3626253.3635427>. doi:10.1145/3626253.3635427.
- [10] S. Wang, J. Mitchell, C. Piech, A large scale rct on effective error messages in cs1, in: Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1, SIGCSE 2024, Association for Computing Machinery, New York, NY, USA, 2024, p. 1395–1401. doi:10.1145/3626252.3630764.
- [11] A. Vadaparty, D. Zingaro, D. H. Smith IV, M. Padala, C. Alvarado, J. Gorson Benario, L. Porter, Cs1-llm: Integrating llms into cs1 instruction, in: Proceedings of the 2024 on Innovation and Technology in Computer Science Education V. 1, ITICSE 2024, Association for Computing Machinery, New York, NY, USA, 2024, p. 297–303. doi:10.1145/3649217.3653584.
- [12] M. Liffiton, B. E. Sheese, J. Savelka, P. Denny, Codehelp: Using large language models with guardrails for scalable support in programming classes, in: Proceedings of the 23rd Koli Calling International Conference on Computing Education Research, Koli Calling ’23, Association for Computing Machinery, New York, NY, USA, 2024. doi:10.1145/3631802.3631830.
- [13] R. Bommasani, D. A. Hudson, E. Adeli, R. Altman, S. Arora, S. von Arx, M. S. Bernstein, J. Bohg, A. Bosselut, E. Brunskill, et al., On the opportunities and risks of foundation models, *arXiv preprint arXiv:2108.07258* (2021).
- [14] C. Koutchme, N. Dainese, S. Sarsa, A. Hellas, J. Leinonen, P. Denny, 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, ITICSE ’24, 2024. doi:10.1145/3649217.3653612.
- [15] Z. Yu, S. Liu, P. Denny, A. Bergen, M. Liut, 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, SIGCSETS 2025, Association for Computing Machinery, New York, NY, USA, 2025, p. 1302–1308. doi:10.1145/3641554.3701844.

- [16] S. Liu, Z. Yu, F. Huang, Y. Bulbulia, A. Bergen, M. Liut, Can small language models with retrieval-augmented generation replace large language models when learning computer science?, in: Proceedings of the 2024 on Innovation and Technology in Computer Science Education V. 1, ITiCSE 2024, Association for Computing Machinery, New York, NY, USA, 2024, p. 388–393. doi:10.1145/3649217.3653554.
- [17] C. Koutchme, N. Dainese, S. Sarsa, A. Hellas, J. Leinonen, S. Ashraf, P. Denny, Evaluating language models for generating and judging programming feedback, in: Proceedings of the 56th ACM Technical Symposium on Computer Science Education V. 1, SIGCSETS 2025, Association for Computing Machinery, New York, NY, USA, 2025, p. 624–630. URL: <https://doi.org/10.1145/3641554.3701791>. doi:10.1145/3641554.3701791.
- [18] B. C. Das, M. H. Amini, Y. Wu, Security and privacy challenges of large language models: A survey, *ACM Computing Surveys* 57 (2025) 1–39.
- [19] Y. Hicke, A. Agarwal, Q. Ma, P. Denny, Ai-ta: Towards an intelligent question-answer teaching assistant using open-source llms, 2023. URL: <https://arxiv.org/abs/2311.02775>. arXiv:2311.02775.
- [20] N. Ashok Kumar, A. Lan, 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, 2024, pp. 108–118.
- [21] J. Woodrow, S. Koyejo, C. Piech, Improving generative ai student feedback: Direct preference optimization with teachers in the loop, https://juliettewoodrow.github.io/paper-hosting/dpo_feedback.pdf, 2025. Accessed: 2025-04-12.
- [22] A. Scarlatos, D. Smith, S. Woodhead, A. Lan, Improving the Validity of Automatically Generated Feedback via Reinforcement Learning, Springer Nature Switzerland, 2024, p. 280–294. doi:10.1007/978-3-031-64302-6_20.
- [23] A. Hellas, J. Leinonen, S. Sarsa, C. Koutchme, L. Kujanpää, J. Sorva, 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, ICER ’23, Association for Computing Machinery, New York, NY, USA, 2023, p. 93–105. URL: <https://doi.org/10.1145/3568813.3600139>. doi:10.1145/3568813.3600139.
- [24] L. Roest, H. Keuning, J. Jeuring, Next-step hint generation for introductory programming using large language models, in: Proceedings of the 26th Australasian Computing Education Conference, ACE ’24, Association for Computing Machinery, New York, NY, USA, 2024, p. 144–153. doi:10.1145/3636243.3636259.
- [25] L. B. Allal, A. Lozhkov, E. Bakouch, G. M. Blázquez, L. Tunstall, A. Piqueres, A. Marafioti, C. Zakka, L. von Werra, T. Wolf, Smollm2 - with great data, comes great performance, 2024.
- [26] A. Dubey, A. Jauhri, A. Pandey, A. Kadian, A. Al-Dahle, A. L. et al., The llama 3 herd of models, 2024. URL: <https://arxiv.org/abs/2407.21783>. arXiv:2407.21783.
- [27] A. de Freitas, J. Coffman, M. de Freitas, J. Wilson, T. Weingart, Falconcode: A multiyear dataset of python code samples from an introductory computer science course, in: Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1, SIGCSE 2023, Association for Computing Machinery, New York, NY, USA, 2023, p. 938–944. doi:10.1145/3545945.3569822.
- [28] C. Zhou, P. Liu, P. Xu, S. Iyer, J. Sun, Y. Mao, X. Ma, A. Efrat, P. Yu, L. Yu, S. Zhang, G. Ghosh, M. Lewis, L. Zettlemoyer, O. Levy, Lima: less is more for alignment, in: Proceedings of the 37th International Conference on Neural Information Processing Systems, NIPS ’23, Curran Associates Inc., Red Hook, NY, USA, 2023.
- [29] L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, A. Ray, et al., Training language models to follow instructions with human feedback, *Advances in neural information processing systems* 35 (2022) 27730–27744.
- [30] D. M. Ziegler, N. Stiennon, J. Wu, T. B. Brown, A. Radford, D. Amodei, P. Christiano, G. Irving, Fine-tuning language models from human preferences, *arXiv preprint arXiv:1909.08593* (2019).
- [31] L. Zheng, W.-L. Chiang, Y. Sheng, S. Zhuang, Z. Wu, Y. Zhuang, Z. Lin, Z. Li, D. Li, E. P. Xing, H. Zhang, J. E. Gonzalez, I. Stoica, Judging llm-as-a-judge with mt-bench and chatbot arena, in:

- Proceedings of the 37th International Conference on Neural Information Processing Systems, NIPS '23, Curran Associates Inc., Red Hook, NY, USA, 2023.
- [32] L. Tunstall, E. Beeching, N. Lambert, N. Rajani, K. Rasul, Y. Belkada, S. Huang, L. von Werra, C. Fourrier, N. Habib, N. Sarrazin, O. Sansevero, A. M. Rush, T. Wolf, Zephyr: Direct distillation of LM alignment, CoRR abs/2310.16944 (2023). URL: <https://doi.org/10.48550/arXiv.2310.16944>. doi:10.48550/ARXIV.2310.16944. arXiv:2310.16944.
 - [33] H. Lee, S. Phatale, H. Mansoor, T. Mesnard, J. Ferret, K. Lu, C. Bishop, E. Hall, V. Carbune, A. Rastogi, S. Prakash, RLAIIF vs. RLHF: scaling reinforcement learning from human feedback with AI feedback, in: Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024, OpenReview.net, 2024. URL: <https://openreview.net/forum?id=uydQ2W41KO>.
 - [34] R. Rafailov, A. Sharma, E. Mitchell, C. D. Manning, S. Ermon, C. Finn, Direct preference optimization: Your language model is secretly a reward model, Advances in Neural Information Processing Systems 36 (2024).
 - [35] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, O. Klimov, Proximal policy optimization algorithms, 2017. URL: <https://arxiv.org/abs/1707.06347>. arXiv:1707.06347.
 - [36] N. Houlsby, A. Giurgiu, S. Jastrzebski, B. Morrone, Q. De Laroussilhe, A. Gesmundo, M. Attariyan, S. Gelly, Parameter-efficient transfer learning for NLP, in: K. Chaudhuri, R. Salakhutdinov (Eds.), Proceedings of the 36th International Conference on Machine Learning, volume 97 of *Proceedings of Machine Learning Research*, PMLR, 2019, pp. 2790–2799. URL: <https://proceedings.mlr.press/v97/houlsby19a.html>.
 - [37] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, W. Chen, Lora: Low-rank adaptation of large language models, in: The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022, OpenReview.net, 2022. URL: <https://openreview.net/forum?id=nZeVKeeFYf9>.
 - [38] C. Koutchme, Training Language Models for Programming Feedback Using Automated Repair Tools, in: N. Wang, G. Rebolledo-Mendez, N. Matsuda, O. C. Santos, V. Dimitrova (Eds.), Artificial Intelligence in Education, Springer Nature Switzerland, Cham, 2023, pp. 830–835.
 - [39] C. Koutchme, S. Sarsa, J. Leinonen, A. Hellas, P. Denny, Automated Program Repair Using Generative Models for Code Infilling, in: N. Wang, G. Rebolledo-Mendez, N. Matsuda, O. C. Santos, V. Dimitrova (Eds.), Artificial Intelligence in Education, Springer Nature Switzerland, Cham, 2023, pp. 798–803.
 - [40] N. Kotalwar, A. Gotovos, A. Singla, Hints-in-browser: Benchmarking language models for programming feedback generation, in: A. Globersons, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. M. Tomczak, C. Zhang (Eds.), Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024, 2024. URL: http://papers.nips.cc/paper_files/paper/2024/hash/34cc2ded6daba59357134c0b9fb06bfe-Abstract-Datasets_and_Benchmarks_Track.html.
 - [41] S. Guo, B. Zhang, T. Liu, T. Liu, M. Khalman, F. Llinares, A. Rame, T. Mesnard, Y. Zhao, B. Piot, J. Ferret, M. Blondel, Direct language model alignment from online ai feedback, 2024. URL: <https://arxiv.org/abs/2402.04792>. arXiv:2402.04792.
 - [42] T. Phung, V.-A. Pădurean, A. Singh, C. Brooks, J. Cambronero, S. Gulwani, A. Singla, G. Soares, 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, LAK '24, Association for Computing Machinery, New York, NY, USA, 2024, p. 12–23. doi:10.1145/3636555.3636846.
 - [43] R. Agarwal, N. Vieillard, Y. Zhou, P. Stanczyk, S. R. Garea, M. Geist, O. Bachem, On-policy distillation of language models: Learning from self-generated mistakes, in: The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024, OpenReview.net, 2024. URL: <https://openreview.net/forum?id=3zKtaqxLhW>.
 - [44] Mistral AI Team, Mistral nemo, <https://mistral.ai/news/mistral-nemo/>, 2024. Accessed: 2024-09-16.
 - [45] S. R. Chowdhury, A. Kini, N. Natarajan, Provably robust dpo: aligning language models with noisy feedback, in: Proceedings of the 41st International Conference on Machine Learning, ICML'24,

- JMLR.org, 2024.
- [46] M. Gheshlaghi Azar, Z. Daniel Guo, B. Piot, R. Munos, M. Rowland, M. Valko, D. Calandriello, A general theoretical paradigm to understand learning from human preferences, in: S. Dasgupta, S. Mandt, Y. Li (Eds.), *Proceedings of The 27th International Conference on Artificial Intelligence and Statistics*, volume 238 of *Proceedings of Machine Learning Research*, PMLR, 2024, pp. 4447–4455. URL: <https://proceedings.mlr.press/v238/gheshlaghi-azar24a.html>.
 - [47] C.-Y. Lin, ROUGE: A package for automatic evaluation of summaries, in: *Text Summarization Branches Out*, Association for Computational Linguistics, Barcelona, Spain, 2004, pp. 74–81. URL: <https://aclanthology.org/W04-1013>.
 - [48] J. Lagus, K. Longi, A. Klami, A. Hellas, Transfer-learning methods in programming course outcome prediction, *ACM Transactions on Computing Education (TOCE)* 18 (2018) 1–18.
 - [49] S. Kar, G. Castellucci, S. Filice, S. Malmasi, O. Rokhlenko, Preventing catastrophic forgetting in continual learning of new natural language tasks, in: *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2022, pp. 3137–3145.
 - [50] C. Koutchme, N. Dainese, A. Hellas, Using program repair as a proxy for language models’ feedback ability in programming education, in: E. Kochmar, M. Bexte, J. Burstein, A. Horbach, R. Laarmann-Quante, A. Tack, V. Yaneva, Z. Yuan (Eds.), *Proceedings of the 19th Workshop on Innovative Use of NLP for Building Educational Applications (BEA 2024)*, Association for Computational Linguistics, Mexico City, Mexico, 2024, pp. 165–181. URL: <https://aclanthology.org/2024.bea-1.15>.
 - [51] Y. Hu, U. Z. Ahmed, S. Mechtaev, B. Leong, A. Roychoudhury, Re-factoring based program repair applied to programming assignments, in: *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, IEEE/ACM, 2019, pp. 388–398.
 - [52] OpenAI, Gpt-4o system card, 2024. URL: <https://arxiv.org/abs/2410.21276>. arXiv:2410.21276.
 - [53] B. Hui, J. Yang, Z. Cui, J. Yang, D. Liu, L. Zhang, T. Liu, J. Zhang, B. Yu, K. Dang, et al., Qwen2.5-coder technical report, arXiv preprint arXiv:2409.12186 (2024).
 - [54] P. Verga, S. Hofstatter, S. Althammer, Y. Su, A. Piktus, A. Arkhangorodsky, M. Xu, N. White, P. Lewis, Replacing judges with juries: Evaluating llm generations with a panel of diverse models, 2024. URL: <https://arxiv.org/abs/2404.18796>. arXiv:2404.18796.
 - [55] Google DeepMind, Gemini 2.0: Our largest and most capable AI model, <https://blog.google/technology/google-deepmind/google-gemini-ai-update-december-2024/>, 2024. Accessed: 2025-04-24.
 - [56] A. S. Thakur, K. Choudhary, V. S. Ramayapally, S. Vaidyanathan, D. Hupkes, Judging the judges: Evaluating alignment and vulnerabilities in llms-as-judges, 2024. URL: <https://arxiv.org/abs/2406.12624>. arXiv:2406.12624.
 - [57] H. Seo, T. Hwang, J. Jung, H. Kang, H. Namgoong, Y. Lee, S. Jung, Large language models as evaluators in education: Verification of feedback consistency and accuracy, *Applied Sciences* 15 (2025) 671. doi:10.3390/app15020671.
 - [58] WebLLM, WebLLM: A web-based language model, <https://webllm.mlc.ai/>, 2024. Accessed: 2024-09-16.
 - [59] Z. Shao, P. Wang, Q. Zhu, R. Xu, J. Song, X. Bi, H. Zhang, M. Zhang, Y. K. Li, Y. Wu, D. Guo, Deepseekmath: Pushing the limits of mathematical reasoning in open language models, 2024. URL: <https://arxiv.org/abs/2402.03300>. arXiv:2402.03300.

A. Dataset

Figure 1 shows the assignments used for the human evaluation.

B. Prompts

Figure 2, figure 3, and figure 4 shows the prompts used in our study.

Assignment 1 - lsn6_lists

Write an algorithm that gets a decimal GPA, APA, and MPA from the user (in that order). You may assume that all inputs are non-negative whole numbers.

It then reports which meritorious list the cadet is on. If the GPA is equal to or above 3.0, the cadet is on the “Dean’s List”, and if the APA is equal to or above 3.0, the cadet is on the “Athletic Director’s List”, and if the MPA is equal to or above 3.0, the cadet is on the “Commandant’s List”. Finally, if the cadet qualifies for all three individual lists, then the cadet is on the “Superintendent’s List”. The algorithm should report all the lists the cadet is on (in the order defined above), unless the cadet is on the Superintendent’s List, in which case, it should report only “Superintendent’s List”.

Assignment 2 - lsn9_imagesize

Write a function that computes the size of an uncompressed image. You will name your function `calculate_size_of_image()`, and it will have three parameters: the width of the image, the height of the image, and the bit depth (i.e., the number of bits per pixel). The function should print the size of the image in kilobytes.

Assignment 3 - lterLogic2_football

In Python, write an algorithm that first asks the user how many football players they wish to enter statistics for and then gets that many yearly passing totals for each player. Output how many of those players had more than 5000 passing yards in a year. Also, your algorithm will output the average yardage per year as well as the minimum yardage entered, in that order. You can assume there is at least one player’s yardage to input.

Assignment 4 - Lists2_movies

Write a Python function called `get_movies` that takes three parameters: * A two-dimensional list containing movie titles and other stats * A rating (e.g., “PG”, “R”) * A run time (in minutes)

Your function should return the number of movies that have the specified rating, and run for at least the number of minutes specified.

Assignment 5 - a3_6_pushups

You have been asked to write a program that analyzes number of pushups done by a group of cadets. Write a program that gets from the user the number of people tested, and gets that many pushup scores (which you may assume are whole numbers) from the user. Your program must print out: * The average number of pushups for the group. * The count of cadets that scored higher than the average.

Figure 1: Assignments used for human annotation of language model responses.

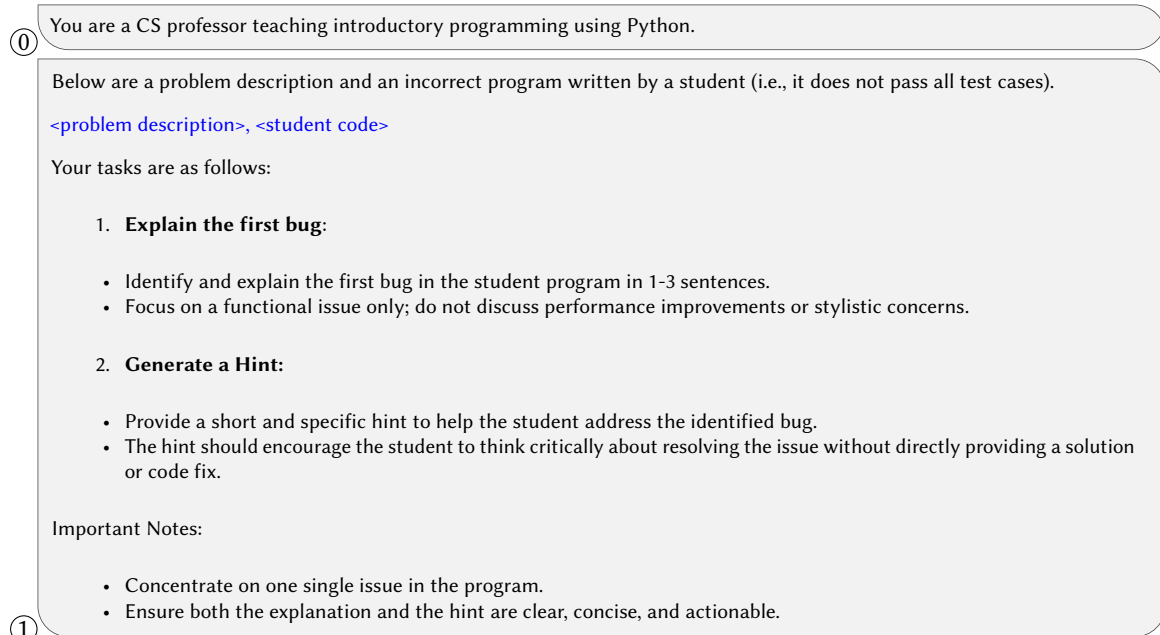


Figure 2: Feedback prompt. Our template for prompting language models to provide feedback. (1) A system prompt specifying the behaviour of the model. (2) A description of the grading task. (3) Information necessary to grade the feedback.

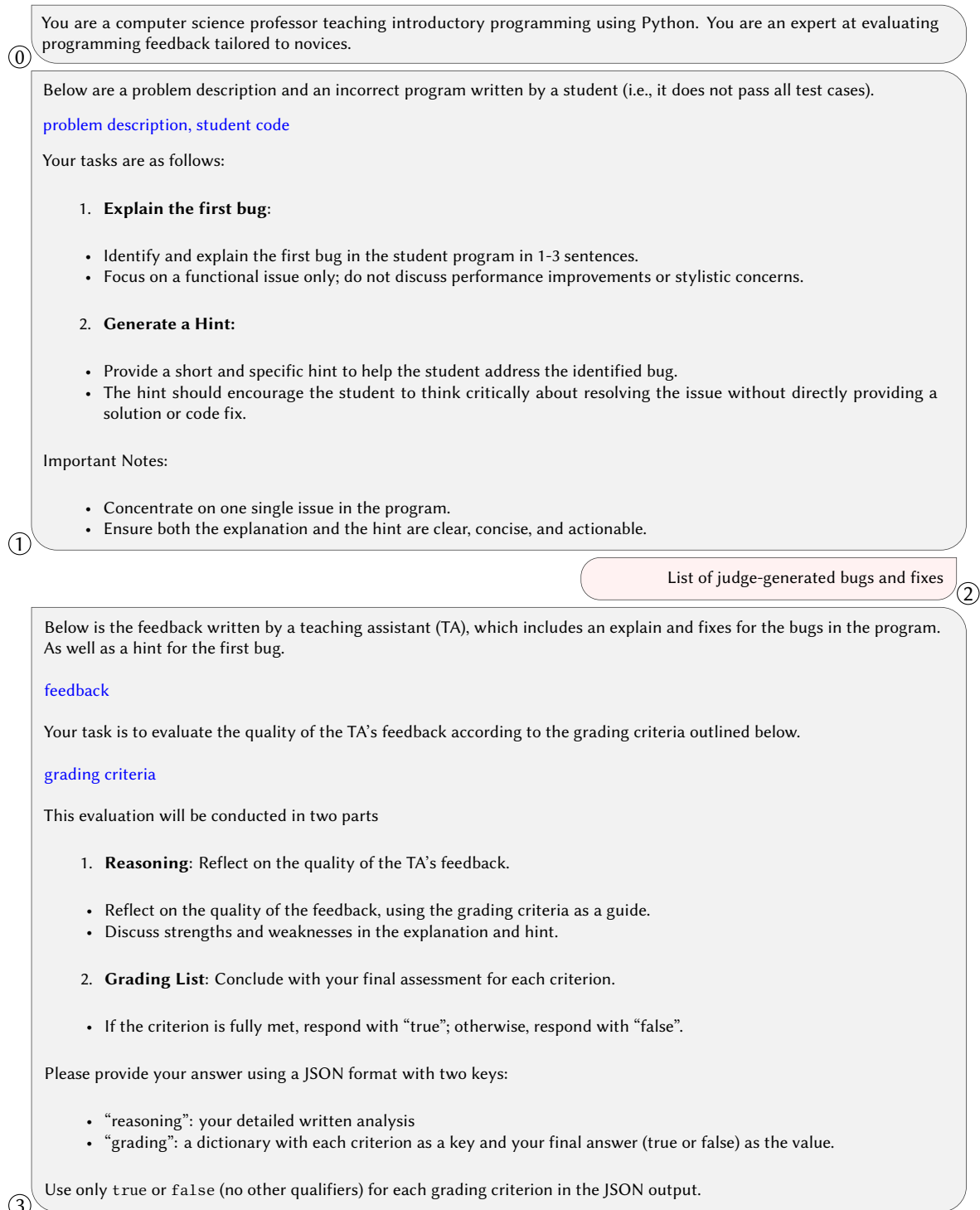


Figure 3: Training judging prompt. We first ask our LLM teacher (GPT-4o-mini) to provide feedback on the incorrect program before asking it to grade the provided one.

① You are a computer science professor teaching introductory programming using Python. You are an expert at evaluating programming feedback tailored to novices.

Below is a problem description and an incorrect program written by a student (i.e., it does not pass all test cases).

[problem description, student code](#)

Below is the feedback written by a teaching assistant (TA), which includes an explain and fixes for the bugs in the program. As well as a hint for the first bug.

[feedback](#)

Your task is to evaluate the quality of the TA's feedback according to the grading criteria outlined below.

[grading criteria](#)

This evaluation will be conducted in two parts

1. **Reasoning:** Reflect on the quality of the TA's feedback.
 - Reflect on the quality of the feedback, using the grading criteria as a guide.
 - Discuss strengths and weaknesses in the explanation and hint.
2. **Grading List:** Conclude with your final assessment for each criterion.
 - If the criterion is fully met, respond with "true"; otherwise, respond with "false".

Please provide your answer using a JSON format with two keys:

- "reasoning": your detailed written analysis
- "grading": a dictionary with each criterion as a key and your final answer (true or false) as the value.

① Use only true or false (no other qualifiers) for each grading criterion in the JSON output.

Figure 4: Evaluation judging prompt. We provide our three LLM judges with a system description describing their role and a description of the judging task. While during training we asked the LLM to first generate its own feedback, we omit this part here as models becoming overly stringent affects ensemble performance.